aktiv computern

Start mit Logo auf dem CPC 464 und 664

Dietrich Senftleben



Dietrich Senftleben

Start mit Logo auf dem CPC464 und 664

HC – Mein Home-Computer

Dr. Dietrich Senftleben

Start mit Logo auf dem CPC 464 und 664

Das kleine Logo-Einmaleins
Grafik · Text · Musik



VOGEL-BUCHVERLAG
WÜRZBURG

DR. DIETRICH SENFTLEBEN

Jahrgang 1941. Studium des Wirtschaftsingenieurwesens und Promotion an der TH Darmstadt. Nach mehreren Berufsjahren in der Groß-EDV Wechsel in den Schuldienst. Unterricht in Mathematik und EDV an kaufmännischen beruflichen Schulen und beruflichen Gymnasien. Tätigkeiten in der Lehrerausbildung, -fortbildung und Bildungsplanung. Gegenwärtiger Schwerpunkt sind Bildschirmtextsysteme unter Einschluß von intelligenten Endgeräten (Personalcomputern).

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Senftleben, Dietrich:
Start mit Logo auf dem CPC 464 und 664:
das kleine Logo-Einmaleins; Grafik, Text, Musik/
Dietrich Senftleben. – 1. Aufl. – Würzburg:
Vogel, 1985.
(HC – Mein Home-Computer)
ISBN 3-8023-0867-0

ISBN 3-8023-0867-0
1. Auflage. 1985

Umschlaggestaltung: Bernd Schröder, Böhl
Herstellung: Alois Erdl KG, Trostberg

Inhaltsverzeichnis

Vorwort

Willkommen bei Logo. Logo ist eine Computersprache, mit der wir spielen, arbeiten und experimentieren können. Dabei gehen wir mit Grafik, Texten, Zahlen, Farben und Tönen um. Über Computer selbst müssen wir nichts wissen. Da Logo eine amerikanische Entwicklung ist, müssen wir ein paar Handvoll englischer Vokabeln lernen. Doch das ist kein Problem, weil wir bereits mit unserem Schulenglisch schon viel zu gut sind, um diese paar Grundwörter nicht zu verstehen. Und wen das dennoch stört, der benutze eben sein Logosystem, um ein ganz persönliches deutsches Logo zu erstellen. Wir können den Computer mit Logo zu tollen Dingen veranlassen, wenn wir nur experimentierfreudig genug sind und ein bißchen durchhalten, um das notwendige kleine Logo-Einmaleins zu lernen. Damit das kleine Einmaleins auch noch Spaß bringt, lernen wir es in Begleitung einer kleinen Schildkröte. Die Schildkröte (englisch: turtle) wird uns weit über die Hälfte des Buches begleiten. Sie versucht, unsere Anweisungen zu befolgen und zeichnet Striche, färbt sie, dreht sich, läuft vor und zurück, löscht Falsches weder aus und ist insgesamt sehr emsig. Wir lernen das Logo-Einmaleins mittels dieser Schildkrötengrafik.

Diese Einführung ist nicht als Trockenkurs gedacht. Vielmehr soll am Rechner mit dem Buch daneben gearbeitet werden. Am Anfang wird alles Schritt für Schritt mit großen Bildschirmfotos begleitet. Man könnte die Fotos fast im Schnellverfahren lesen. Doch bitte etwas Geduld. Ein wenig Text sollte schon gelesen werden. Auch die Forschungsaufträge am Ende jeder Lektion sind nicht immer einfach. Lösungen muß jeder selbst finden. Wer es dennoch eilig hat, sollte prüfen, ob für ihn die Kurzform der Lektion am Ende jeder Lektion ausreicht. Vor dieser Kurzform der Lektion werden die neu behandelten Logovokabeln aufgelistet und ebenso neue Sondertasten mit ihren Funktionen. Nach dem kleinen Einmaleins beschäftigen wir uns noch

mit Texten und Musik. Gleichzeitig lernen wir ganz wesentliche Vokabeln von Logo hinzu, mit denen wir bereits Teile des großen Einmaleins beherrschen lernen. In den jeweiligen Forschungsaufträgen werden weitere Aspekte von Logo vorgestellt, die man nur bei wirklichem Interesse erforschen sollte.

Der Anhang bietet eine Kurzgrammatik, die unter anderem auch die wesentlichen Anfängerfehler noch einmal aufzählt. Alle Schneider-Logo-Vokabeln werden summarisch aufgeführt. Die vorliegende Logoversion stammt von Digital Research und heißt daher DR.Logo. Das ursprüngliche DR.Logo ist für die Firma Amstrad und damit Schneider in einer verkleinerten Version für den CPC 464 und 664 implementiert worden. Wer bereits andere Versionen kennt, wird manche Logovokabel vermissen. Auch muß man sich an die Kleinschreibung gewöhnen. Weiterhin ist eine Handvoll CP/M-Befehle zu lernen, um Logoprogrammdateien zu listen, löschen und umzubenennen. Dies alles wird hier gezeigt. Damit bietet sich als Fortführung das Buch «Programmieren mit Logo» vom Vogel-Buchverlag an.

Man findet auch nützliche Hilfsprogramme im Anhang, die manche Logovokabel anderer Logoversionen anbieten und damit Schneider-Logo erweitern. Logo lebt von den Ideen des Benutzers, wenn er erst einmal die Grundlagen beherrscht. Starten wir und lernen das kleine Logo-Einmaleins.

Heusenstamm Dietrich Senftleben

1
Das kleine Logo-Einmaleins mittels Grafik

Lektion 1: Willkommen bei Schneider-Logo

Logo wird gestartet, indem wir es von der Logo-Systemdiskette in unseren Computer laden. Gehen wir davon aus, daß der Computer richtig installiert worden ist. Der Strom ist nicht eingeschaltet.

Nun schalten wir alle Geräte ein, als erstes das Diskettenlaufwerk. Haben wir alles richtig gemacht, muß der nachfolgender Text auf dem Schirm erscheinen.

```
Schneider 64K Microcomputer (v1)

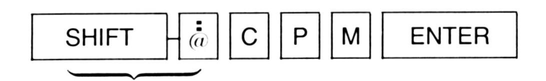
c 1984 Amstrad Consumer Electronics plc
           and Locomotive Software Ltd.

BASIC 1.0

READY
■
```

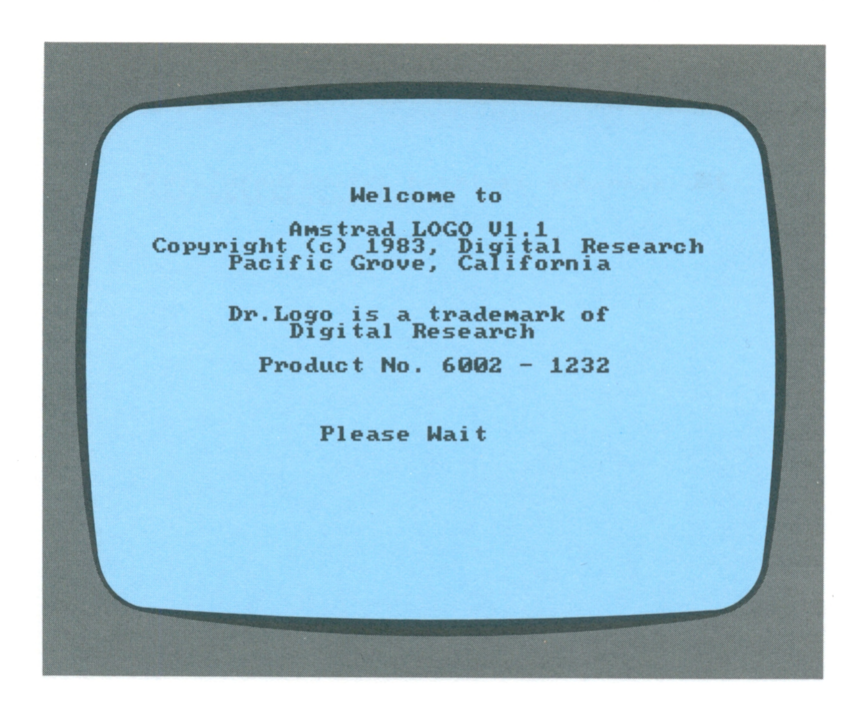
Schieben wir die Logo-Systemdiskette in das Laufwerk A. Achten wir darauf, daß die Diskette korrekt hineingeschoben worden ist.

Jetzt müssen wir folgende Tasten der Reihe nach eintippen.



Diese zwei Tasten gleichzeitig drücken, zuerst SHIFT niedergedrückt halten und dann die zweite Taste drücken.

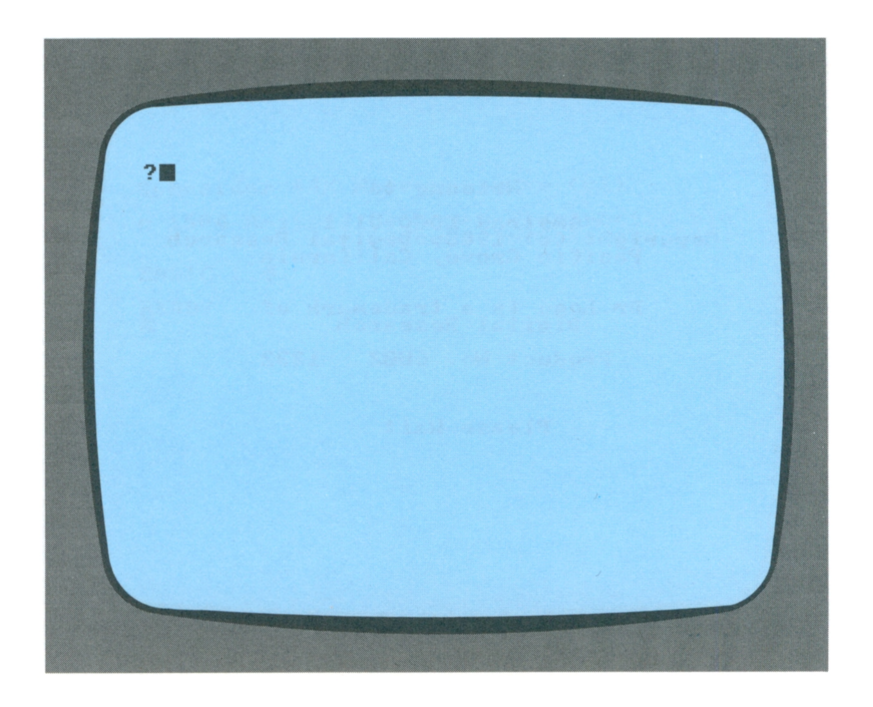
Nach einigen Sekunden erscheint die nachfolgende Begrüßung:



Nach einigen Augenblicken verschwindet die Begrüßung, und der Bildschirm wird gelöscht.

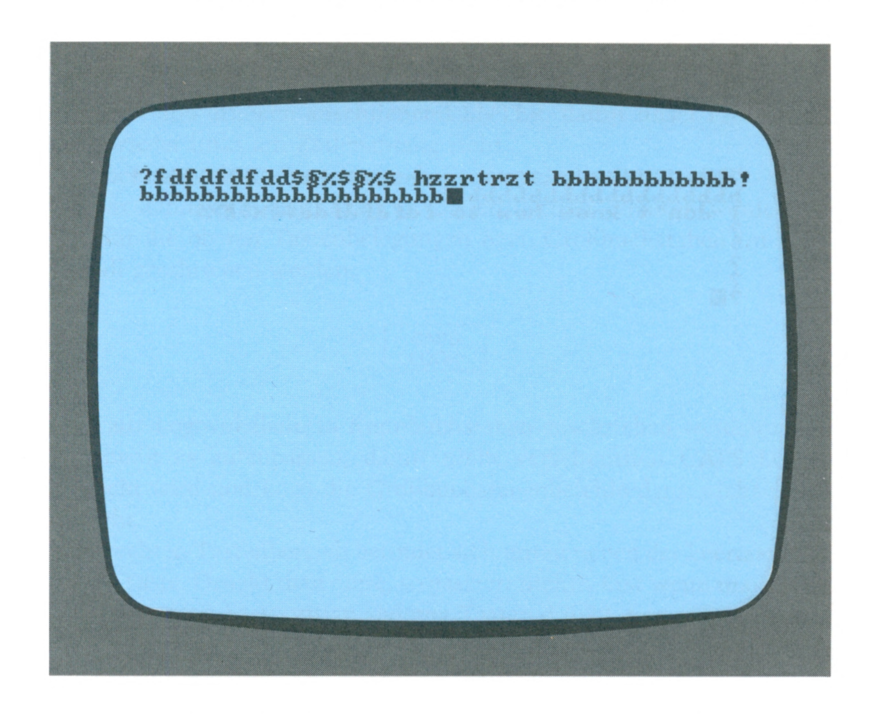
In der ersten Zeile steht ganz links ein Fragezeichen. Dieses Fragezeichen bedeutet, daß der Computer bereit ist, irgendwelche Anweisungen und Eingaben von uns auszuführen. Man könnte fast meinen, daß der Computer uns in Kurzform fragt, was wir denn bitte möchten. Dieses Aufforderungszeichen nennt man auch Promptzeichen.

Neben dem Promptzeichen befindet sich ein Rechteck. Solch ein Zeichen wird auch als Cursor bezeichnet. Der Cursor zeigt auf dem Bildschirm immer die Stelle auf, an der man sich befindet und an der das nächste einzutippende Zeichen erscheinen wird.



Tippen wir einmal irgendwelche Buchstaben ein. Halten wir als nächstes den Buchstaben «B» länger gedrückt. Genau! Viele Bs werden jetzt ausgelöst und tanzen auf der Bildschirmzeile entlang. Setzen wir das bis in die Folgezeile fort.

Sind eingetippte Zeilen länger als eine Bildschirmzeile, so werden die Zeichen in den nächsten Zeilen abgebildet. An der letzten Bildschirmstelle wird ein Fortsetzungszeichen ausgegeben. Dieses Zeichen zeigt uns also, daß keine neue und isolierte Zeile folgt, sondern dieses zur oberen Zeile dazugehört.

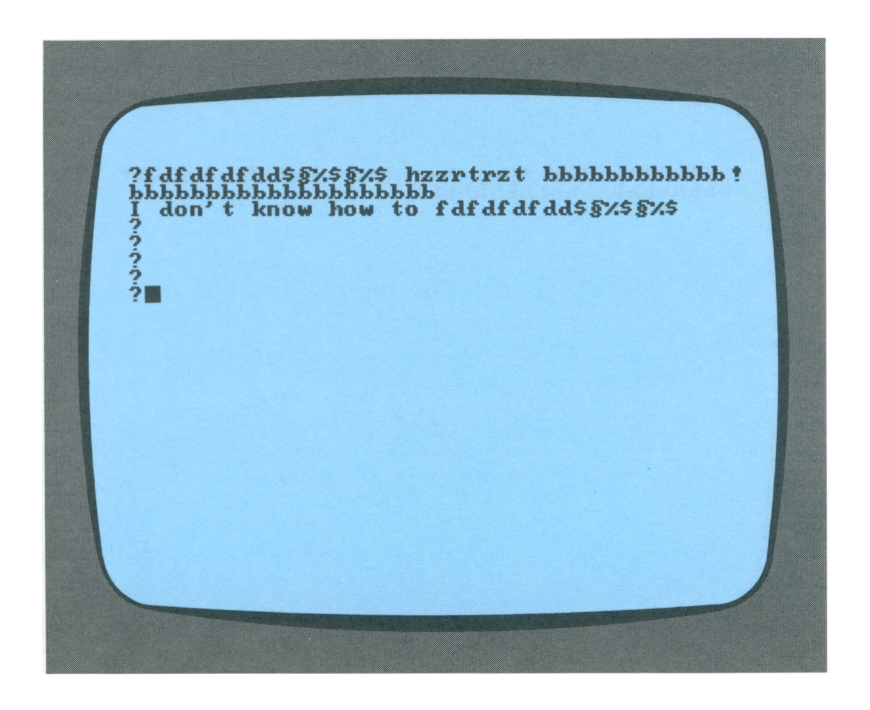


Nachfolgend sollen einige Tasten und deren Bedeutung besprochen werden. Rechts auf der Tastatur befindet sich die schon benutzte ENTER-Taste.

Drücken wir sie einmal. Nicht darum kümmern, was der Computer jetzt auf dem Schirm produziert. Drücken wir noch einige Male die ENTER-Taste. Nach jedem ENTER springt das Promptzeichen eine Zeile tiefer. Mit der ENTER-Taste werden immer Eingaben beendet, die dann der Computer ausführt. Haben wir Unsinn eingetippt – wie oben die Buchstaben –, meckert er natürlich. Über solche Fehlermeldungen erfahren wir später mehr.

Sehr nützlich ist die Taste zum Löschen eines falsch eingetippten Zeichens. Sie befindet sich rechts in der oberen Reihe.

DEL



Mit dieser Taste löschen wir das Zeichen links vom Cursor. Probieren wir's gleich aus. Drücken wir eine Zeitlang irgendeine Buchstabentaste, und anschließend löschen wir alle Zeichen wieder aus. DEL ist die Abkürzung von delete (löschen).

Ab Lektion 2 werden Leerzeichen eine wichtige Rolle spielen. Leerzeichen werden mit der großen langen Taste ganz unten auf der Tastatur erzeugt. Probieren wir's aus! Der Cursor wandert auf der Zeile nach rechts. Drücken wir zum Abschluß einige Male die ENTER-Taste.

Neben der Leertaste befinden sich links und rechts die sogenannten SHIFT-Tasten (Umschalttasten).

SHIFT

Halten wir diese Taste gedrückt und tippen beispielsweise die Zahlentasten, so werden die oberen Tastenbelegungen angesprochen. Probieren wir's und drücken anschließend ENTER, damit unser Promptzeichen wieder links am Zeilenanfang steht.

Neben der SHIFT-Taste rechts befindet sich die CAPS-LOCK-Taste. Mit ihr kann die SHIFT-Taste in Dauerfunktion gehalten werden, so daß nur die oberen Tastenbelegungen bzw. Großbuchstaben im Normalfall geschrieben werden.

CAPS LOOK

Weil alle Logovokabeln in Kleinbuchstaben geschrieben werden müssen, heißt es aufzupassen, damit nicht versehentlich CAPS LOCK gedrückt wird und Logo die Großbuchstabenbefehle dann nicht mehr versteht.

Im rechten Tastenbereich befindet sich unten neben der Leertaste die CTRL-Taste. Drückt man diese, so passiert nichts. Erst wenn man diese gleichzeitig mit bestimmten anderen Tasten drückt, treten Effekte ein.

CTRL

Probieren wir das jetzt aus, indem wir die CTRL-Taste gedrückt halten und dann die G-Taste drücken. Auf dem Schirm erscheint dann nach jedem CTRL-G ein «stopped!».

CTRL	G

Läuft also etwas nicht in unserem Sinn, können wir damit irgendwelche ablaufenden Logoprogramme stoppen.

ESC

Die gleiche Wirkung wie CTRL-G hat die ESC-Taste. ESC kommt von escape und bedeutet flüchten, entkommen.

Logo kennt einige Zeichen, die im deutschen Zeichensatz nicht standardmäßig vorkommen. Dies sind die Zeichen «[», «]» und «/».

Um das Logosystem zu verlassen, müssen wir nicht das Gerät aus- und wiederanschalten. Wir müssen uns nur mit bye ⟨ENTER⟩ verabschieden. Um Logo erneut zu laden, müssen wir den Computer rücksetzen (reset) durch Drücken der Tasten

SHIFT – CTRL – ESC

und Logo wie auf Seite 10 beschrieben laden.

Bevor wir zur Lektion 2 übergehen, wollen wir den Bildschirm noch löschen. Das geschieht durch Eintippen der Buchstaben ct und anschließendes Drücken der ENTER-Taste.

Im Folgekapitel beschäftigen wir uns mit Grafik und lassen durch eine Schildkröte Striche zeichnen. Im Grafikmodus wird uns die Stelle auf dem Zeichenschirm nicht durch einen Cursor, sondern durch eine kleine Schildkröte angezeigt. Sie wird durch ein kleines Dreieck symbolisiert.

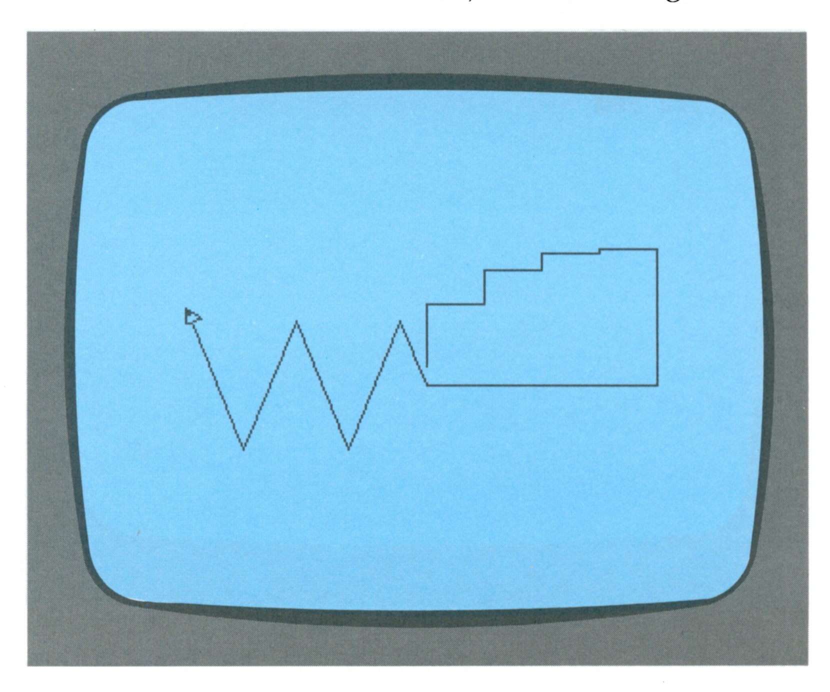
Neue Logo-Vokabeln

ct	Löschtaste
ENTER-Taste	CAPS-LOCK-Taste
	CTRL-Taste
	SHIFT-Taste
	DEL-Taste
	ESC-Taste

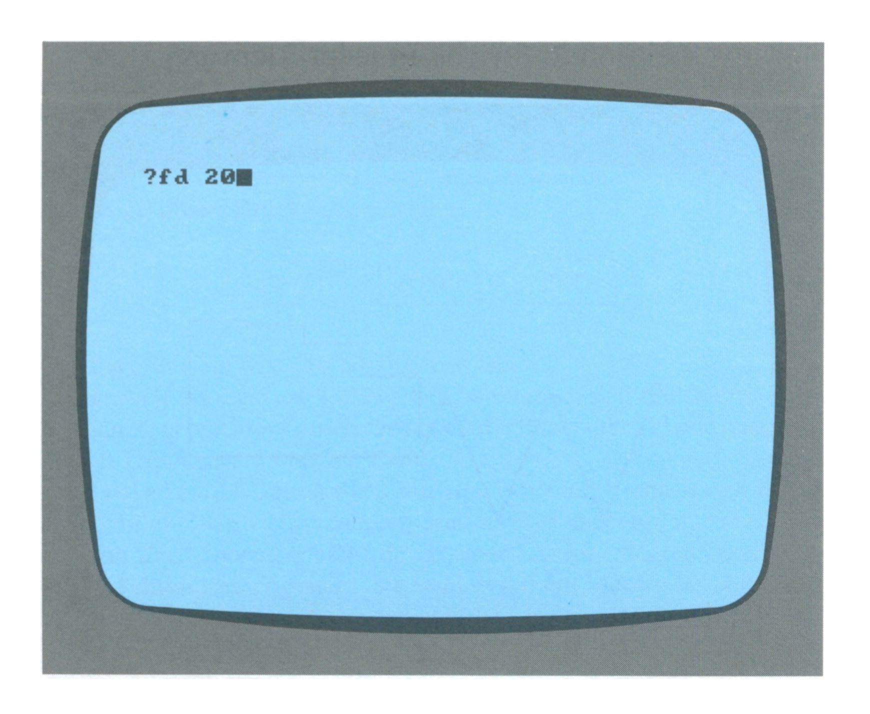
Kurzform der Lektion

Wir haben Schneider-Logo gestartet und den Rechner betriebsklar gemacht. Das Promptzeichen und der Cursor werden vorgestellt. Zeilen, die mehr als 40 Zeichen je Zeile haben, erkennt man am Fortsetzungszeichen in der letzten Stelle der Bildschirmzeile. Die ENTER-, Zeichenlösch-, DEL-, SHIFT-, CTRL-, ESC- und CAPS-LOCK-Taste werden beschrieben. Der Computer meckert das erstemal über eingetippten Unsinn. Mit ENTER und CTRL-G beziehungsweise ESC kommt man aus den meisten Fehlersituationen heraus.

Lektion 2: Striche in jeder Richtung

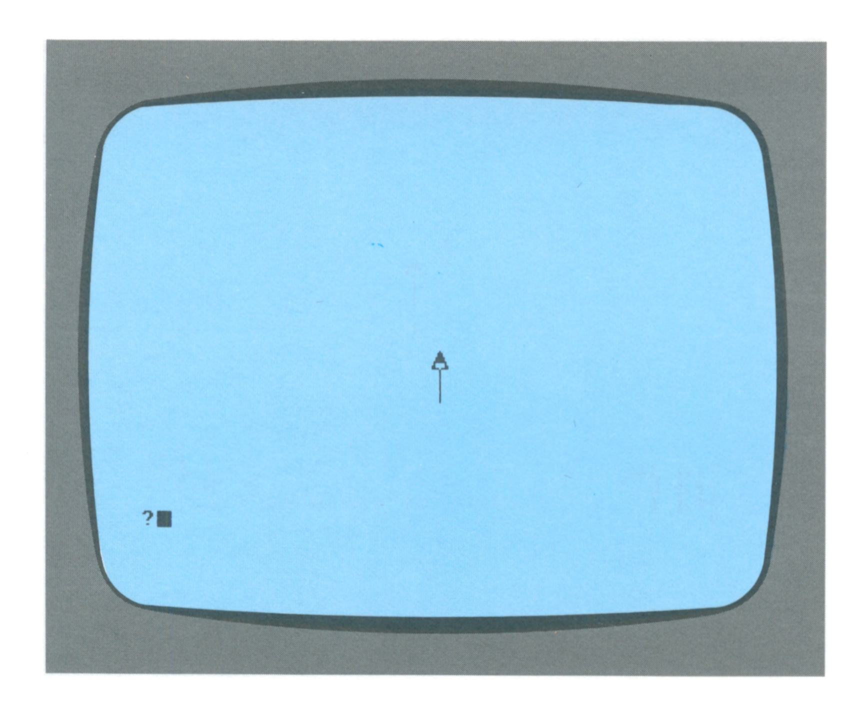


Ohne lange Vorrede wollen wir mit unserem elektronischen Bleistift auf dem Bildschirm stricheln. Am Ende von Lektion 2 ist obige Strichelei entstanden.



Ziehen wir als erstes einfach einen Strich nach oben. Geben wir dazu einmal fd 20 ⟨ENTER⟩ ein (fd ist die Abkürzung von forward und heißt vorwärts). Das Wort ENTER in spitzen Klammern bedeutet, daß wir die entsprechende ENTER-Taste drücken müssen. Um Mißverständnisse zu vermeiden, sind alle zu drückenden Tasten noch einmal gezeigt:

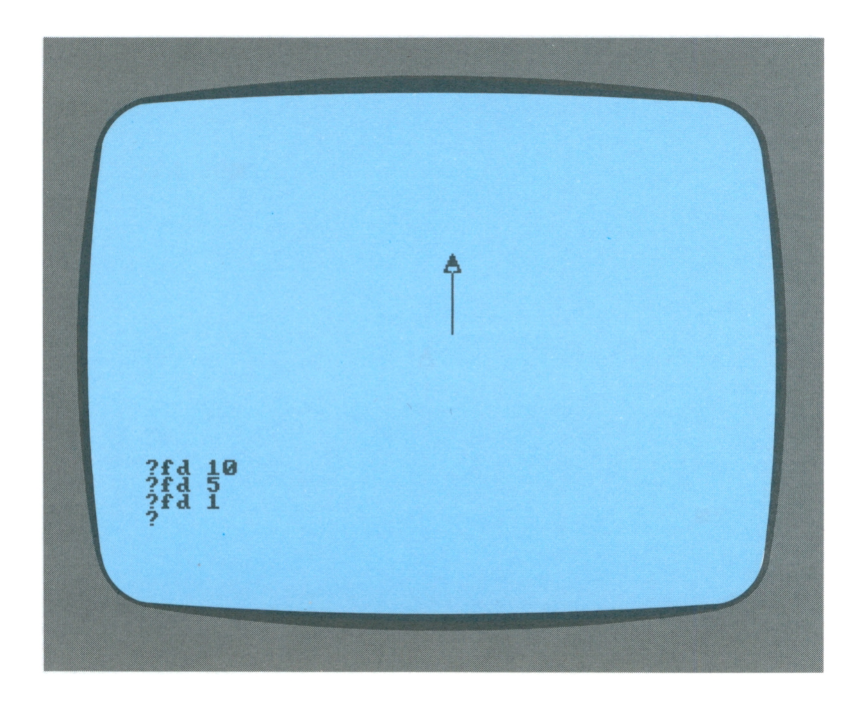
[F] [D] [LEERTASTE] [2] [0] [ENTER]



Mit dem Drücken der ENTER-Taste wird immer eine Anweisung beendet. Jetzt erst kann der Computer die Anweisung fd 20 ausführen.

Am Ende der abgebildeten Strecke sehen wir ein kleines Dreieck. Dieses Symbol soll eine Schildkröte kennzeichnen. Die Schildkröte zeigt uns immer die Richtung an, in die sie sich bei einem Vorwärtsbefehl bewegen würde. Die obere Spitze des Dreiecks zeigt uns die jeweilige Richtung an. Der Vorwärtsbefehl fd läßt also immer einen Strich in der angezeigten Richtung entstehen. Die Zahl nach dem Vorwärtsbefehl gibt die Anzahl der zu machenden Schritte vor. Die Zahl ist also ein Maß für die Länge der gewünschten Strecke.

Der Bildschirm ist zweigeteilt. Im unteren Teil werden die letzten vier eingetippten Zeilen mitprotokolliert. In der letzten Zeile erscheint das aktuell Eingetippte.

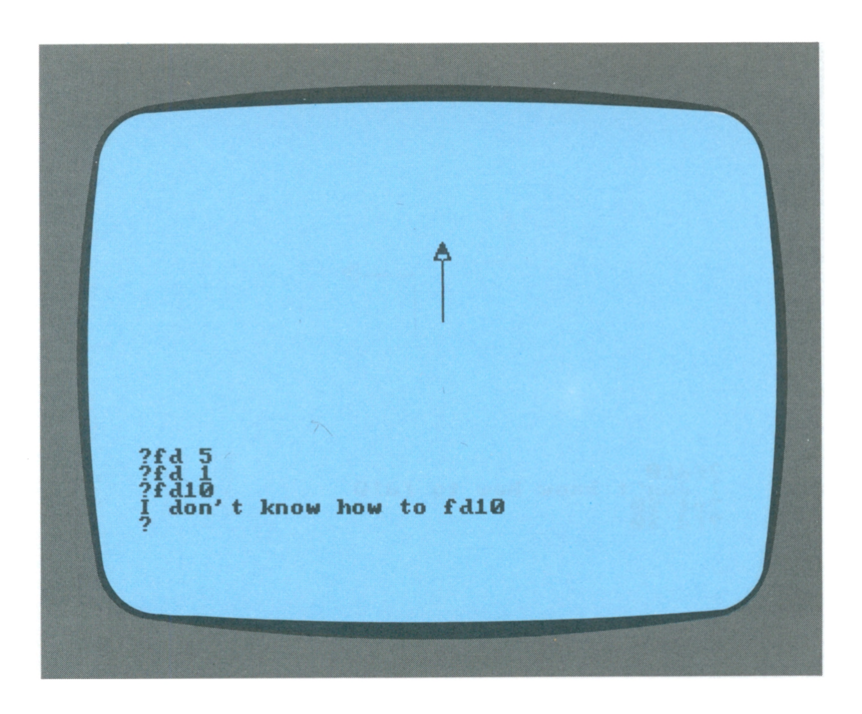


Lassen wir jetzt einmal die Schildkröte zehn, dann fünf Schritte und auch gerade mal einen Schritt vorwärts machen.

```
fd 10  ⟨ENTER⟩
fd  5  ⟨ENTER⟩
fd  1  ⟨ENTER⟩
```

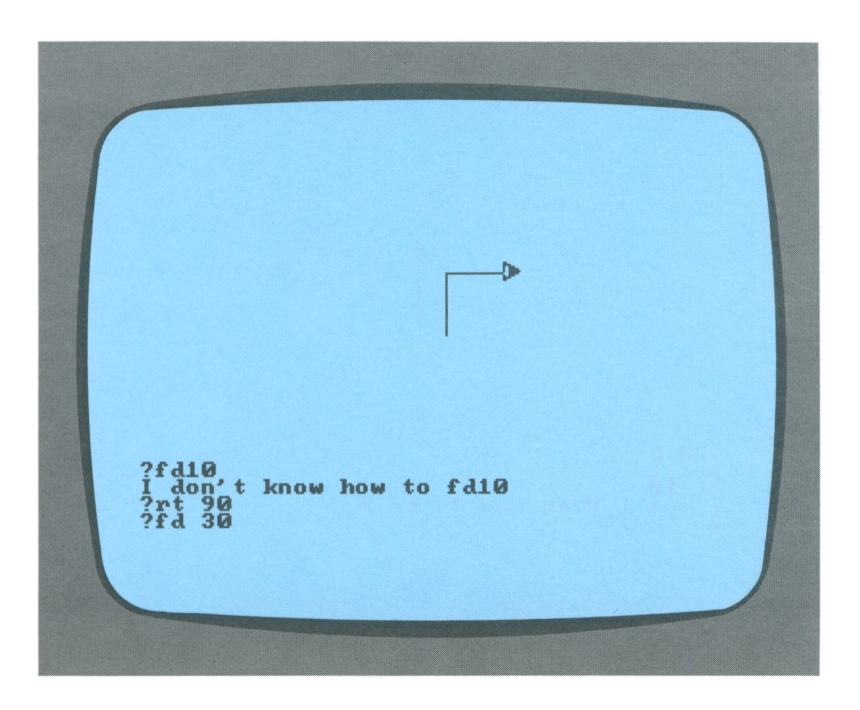
Genau so ist es richtig. Die Abkürzung für den Vorwärtsbefehl lautet fd. Zwischen dem Befehl und der nachfolgenden Zahleneingabe muß ein Leerzeichen stehen. An dem Leerzeichen erkennt der Computer, daß der Befehl zu Ende ist und jetzt weiteres kommt. Solch ein Leerzeichen nennt man auch Trennzeichen oder Trenner. Die Trennzeichen ermöglichen also dem Computer das Erkennen der einzelnen Bestandteile eingegebener Zeilen.

Was passiert eigentlich, wenn wir so ein Leerzeichen vergessen? Probieren wir's doch gleich aus.



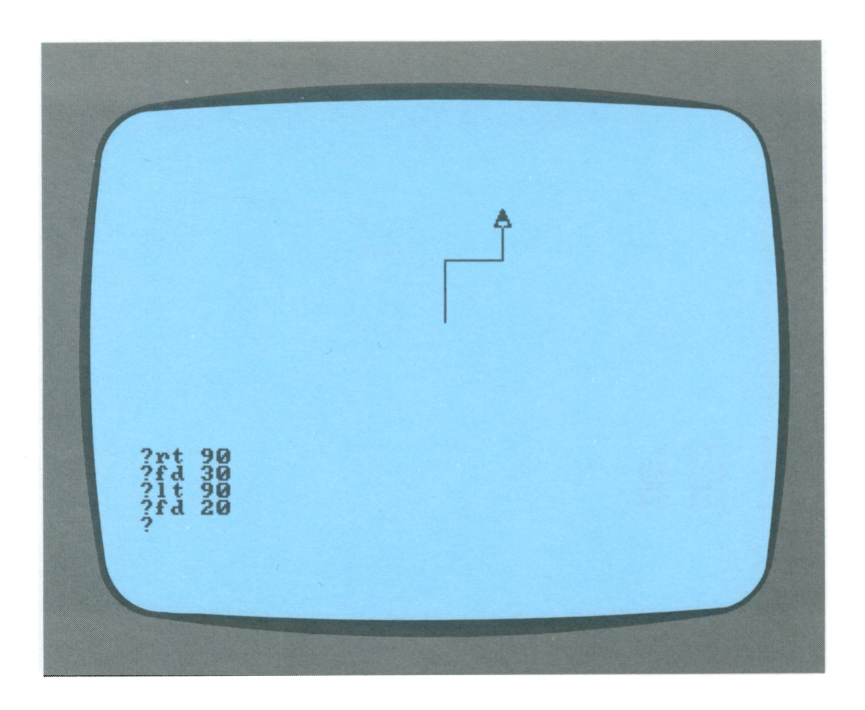
Die Schildkröte sollte weitere zehn Schritte vorwärtsgehen. Doch statt fd 10 haben wir fd10 eingegeben. Der Computer kennt fd10 nicht. Er meldet, daß er nicht weiß, wie fd10 gemacht wird. Da das Trennzeichen fehlt, hält der Computer fd 10 für einen Befehl an sich, den es aber leider nicht gibt, beziehungsweise wir haben es ihm nicht erklärt. Auf dem Bildschirm hat sich nichts verändert.

Solange wir das Korrigieren von Tippfehlern nicht gelernt haben, müssen wir die Anweisungszeile erneut eintippen. Wenn nur ein vorangegangenes Zeichen zu korrigieren ist und wir noch nicht die ENTER-Taste gedrückt haben, können wir mit der Taste DEL das Zeichen links vom Blinkzeichen löschen. Dies ist die Löschtaste oben rechts. Vergleiche Lektion 1. Dann tippen wir einfach das richtige Zeichen ein.

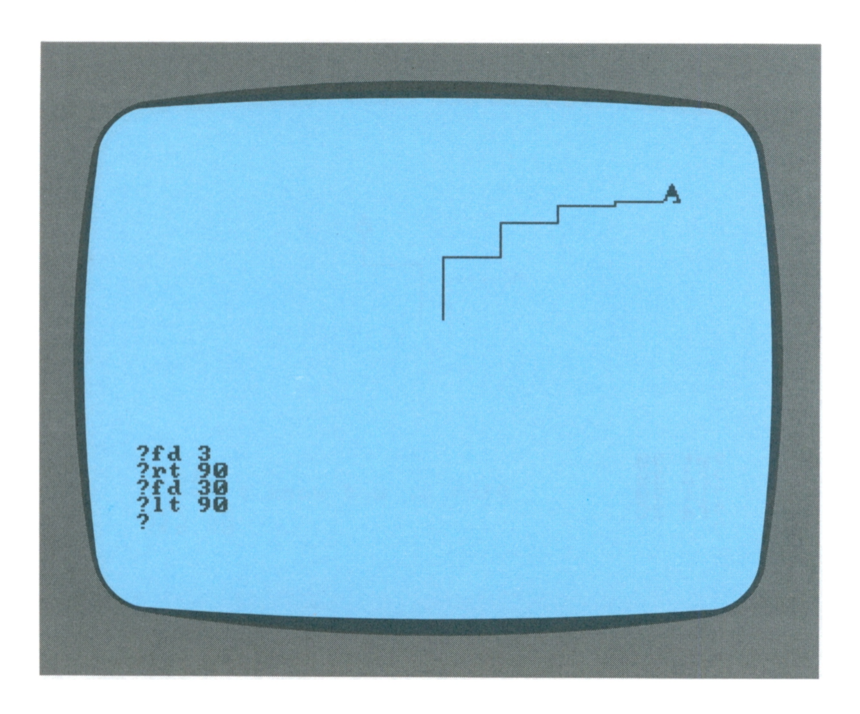


Wollen wir unsere Schildkröte in eine andere Richtung laufen lassen oder einen Strich nach links oder rechts machen, müssen wir zuerst die gewünschte Richtung angeben. Die Dreiecksspitze muß daher als erstes in die gewünschte Richtung gedreht werden. Lassen wir unsere Schildkröte einmal eine Treppe hinaufkriechen. Wir drehen das Symbol um 90 Grad nach rechts und lassen dann die Schildkröte 30 Schritte vorwärtslaufen (right heißt Englisch rechts, rt ist die Abkürzung).

```
rt 90
fd 30
```



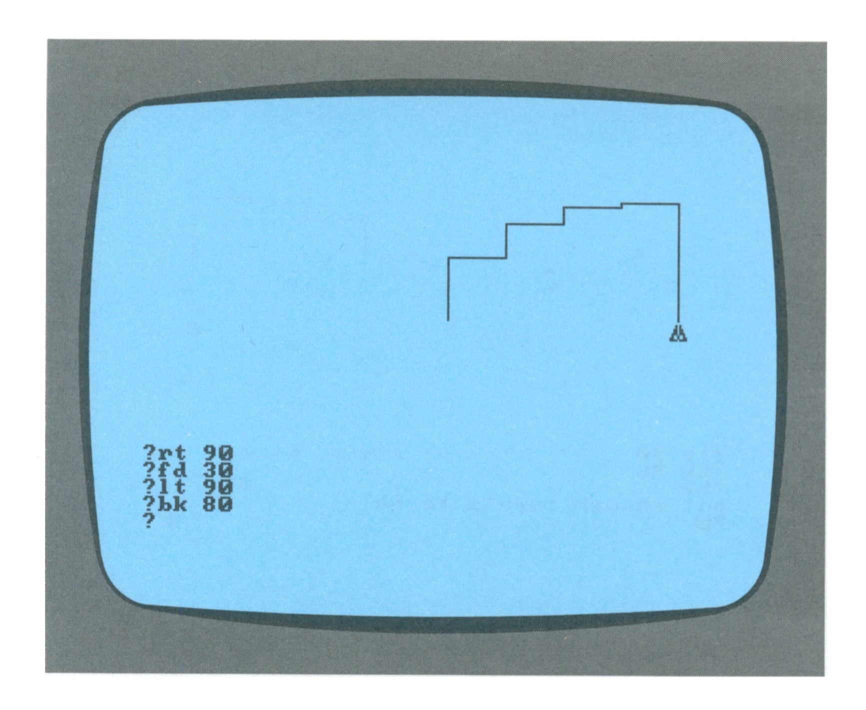
Um die nächste Stufe zu erklimmen, muß die Schildkröte die Richtung ändern. Sie muß wieder nach oben kriechen. Die Linksdrehung erfolgt mit dem Befehl lt (left heißt links, abgekürzt lt). Wir drehen das Dreieck um 90 Grad zurück und gehen dann 20 Schritte nach oben.



Zeichnen wir nun bis in die rechte obere Bildschirmecke eine Treppe, deren Stufen immer kleiner werden sollen.

```
rt  90  ⟨ENTER⟩
fd  30  ⟨ENTER⟩
lt  90  ⟨ENTER⟩
fd  10  ⟨ENTER⟩
rt  90  ⟨ENTER⟩
fd  30  ⟨ENTER⟩
lt  90  ⟨ENTER⟩
fd   3  ⟨ENTER⟩
rt  90  ⟨ENTER⟩
fd  30  ⟨ENTER⟩
lt  90  ⟨ENTER⟩
```

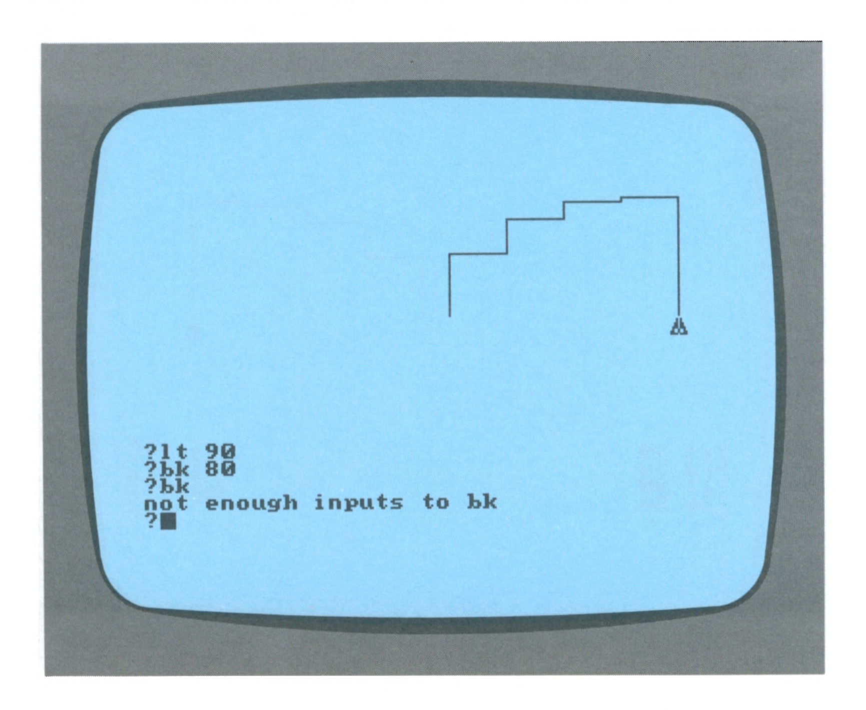
Das Dreieck zeigt nach der letzten Linksdrehung nach oben.



Die Schildkröte kann auch rückwärts laufen. Hierzu müssen wir den Befehl bk (back heißt zurück, abgekürzt bk) benutzen. Natürlich muß die Anzahl der Schritte zusätzlich angegeben werden. Machen wir also einen Strich nach unten bis zur Mittellinie des Schirms.

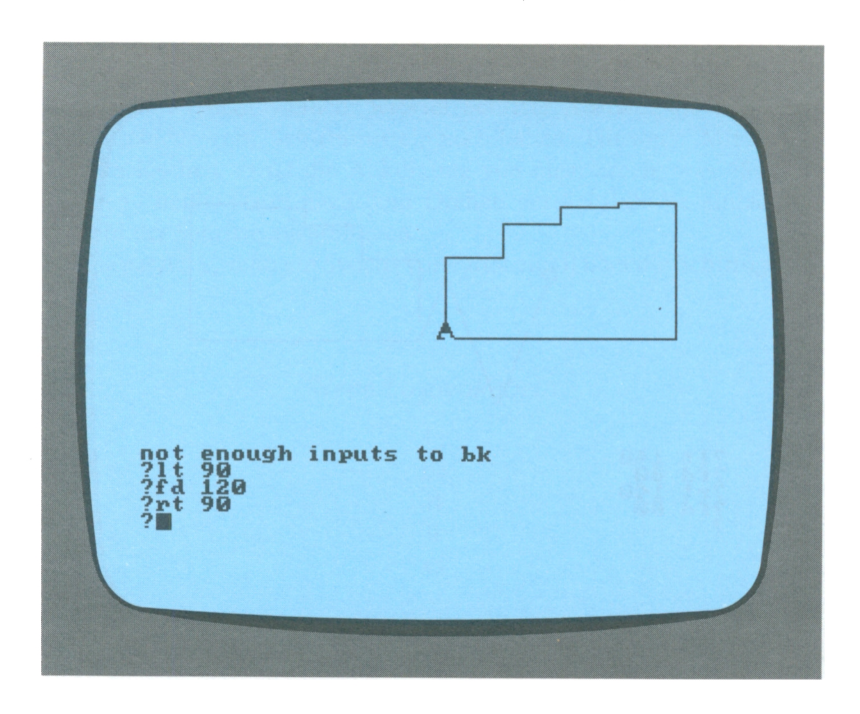
bk 80 ⟨ENTER⟩

Auch beim Rückwärtsbefehl müssen wir wie bei der Vorwärtsbewegung vorher immer die Richtung festlegen. Weil die Schildkröte in unserem Beispiel schon die vorgesehene Richtung hat, entfällt eine Drehung.

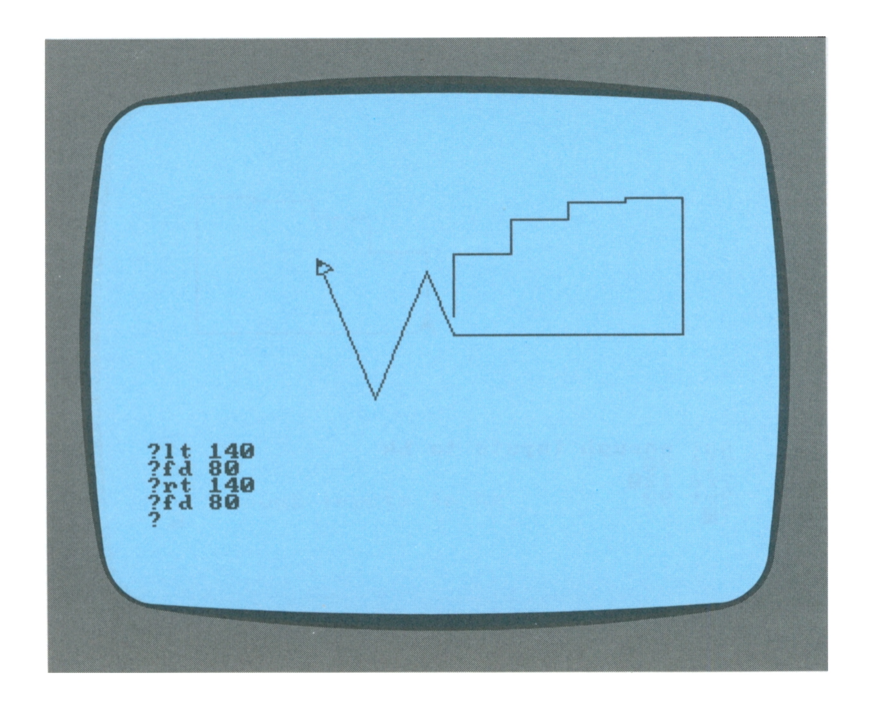


Was passiert eigentlich, wenn wir das Angeben der Schritte vergessen? Probieren wir's aus!

Wir erhalten eine Fehlermeldung. Die Meldung besagt, daß bk noch weitere Eingaben benötigt, daß heißt, die Anzahl der Schritte fehlt. Das Promptzeichen erscheint in der nächsten Zeile, und wir könnten den Befehl richtig eingeben. Wegen der fehlerhaften Anweisung hat sich natürlich die Zeichnung nicht verändert.



Die Schildkröte soll von der jetzigen Position zu ihrer Ausgangslage zurückbeordert werden. Hierzu geben wir einfach die abgebildeten Befehle ein.



Zum Abschluß wollen wir noch eine Sägezahnlinie bis zum linken Rand erzeugen. Die erste Zacke entsteht durch folgende Befehle:

```
lt 20
fd 40
lt 140
fd 80
rt 140
```

Die restlichen Zacken entstehen durch entsprechende Wiederholung der letzten drei Befehle.

Forschungsaufträge

Es soll untersucht werden, wieviel Schritte Abstand die jeweiligen Schirmränder vom Schirmmittelpunkt haben. Hinweis: Mit cs den Schirm löschen. cs ist die Abkürzung von clearscreen, was «Schirmlöschen» heißt. Dann nach dem Drehen der Schildkröte einfach die Anzahl der Schritte ausprobieren.

Was passiert, wenn ein Strich (Vorwärtsbefehl) über die Schirmgrenze hinausgeht?

Neue Logovokabeln

fd
bk
rt
lt
cs

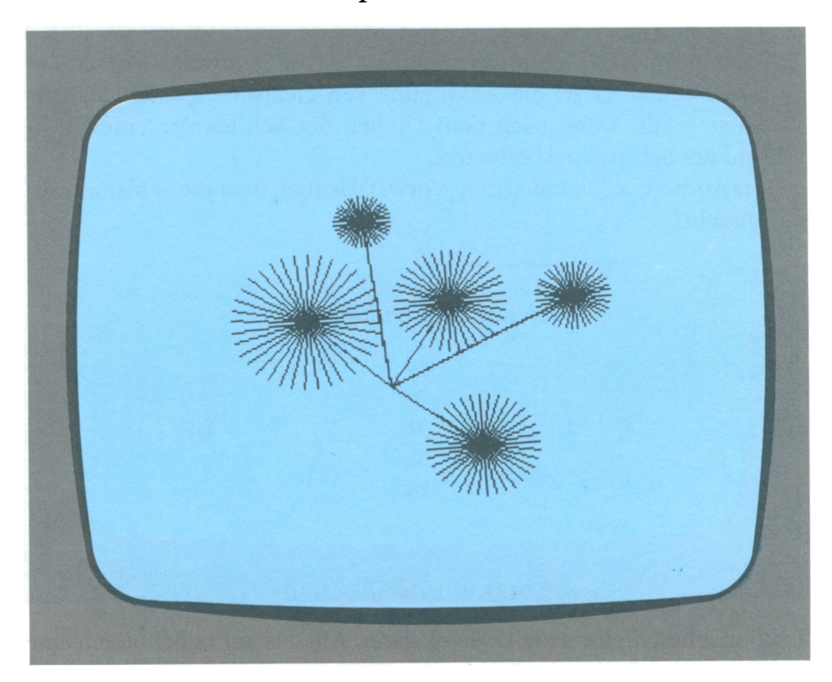
Kurzform der Lektion

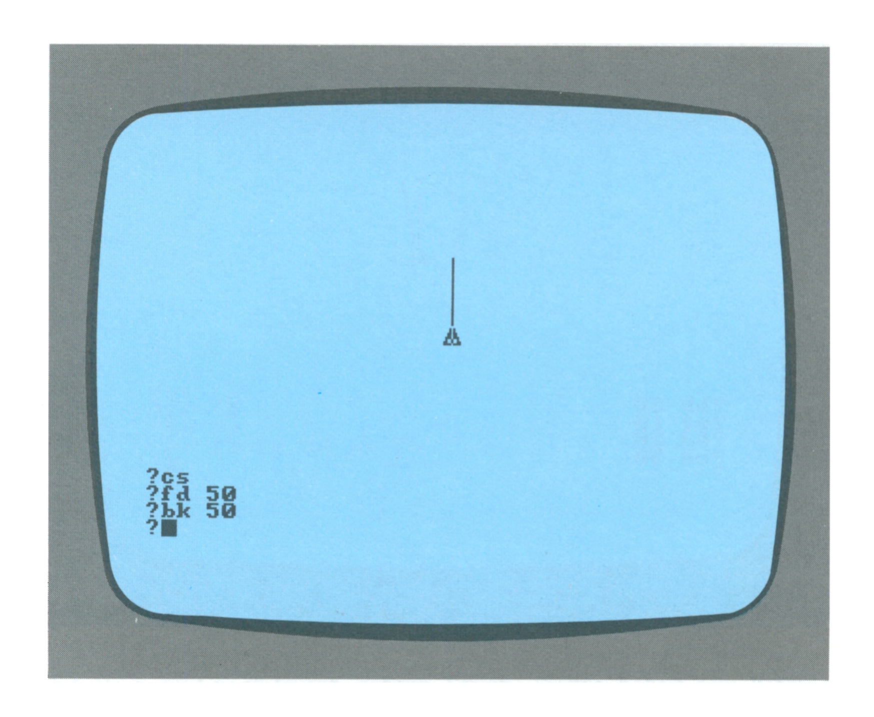
Lektion 2 beschreibt neue Logovokabeln. Alle bis auf cs benötigen eine Zahlenangabe. Striche in die jeweils gewünschte Richtung macht man, indem die Schildkröte in die gewünschte Richtung gedreht wird und mit fd oder bk ein entsprechender Strich gemacht wird.

Wichtig ist die Eingabe des Leerzeichens zwischen Befehl und Zahl. Leerzeichen haben Trennereigenschaft, mit dessen Hilfe der Computer Zeilen und seine Bestandteile erkennen kann.

Die ENTER-Taste muß immer am Ende einer Eingabezeile gedrückt werden. Zwei häufige Fehler und deren Folgen sind vorgeführt worden.

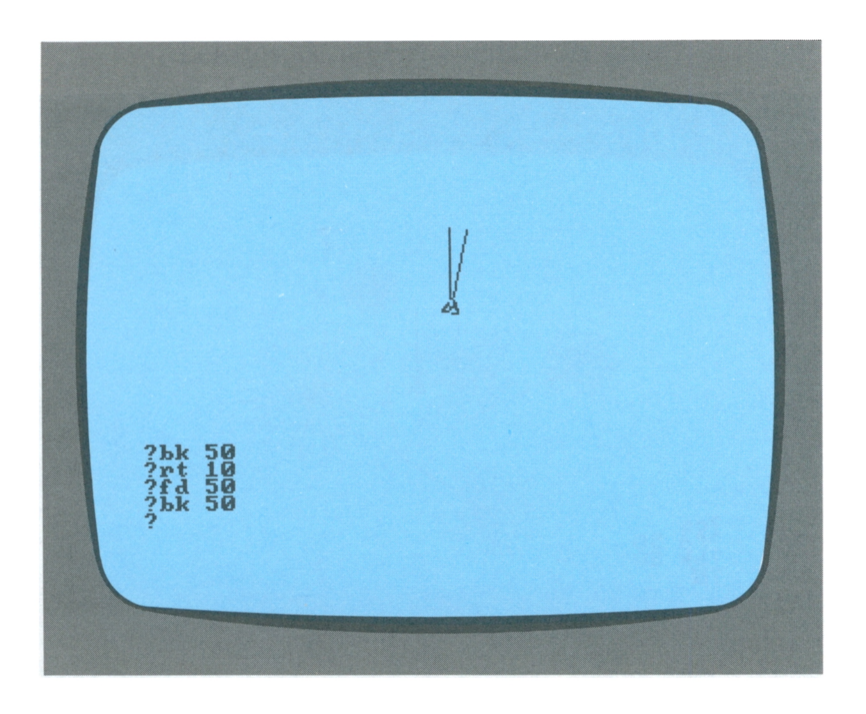
Lektion 3: repeat wiederholt alles





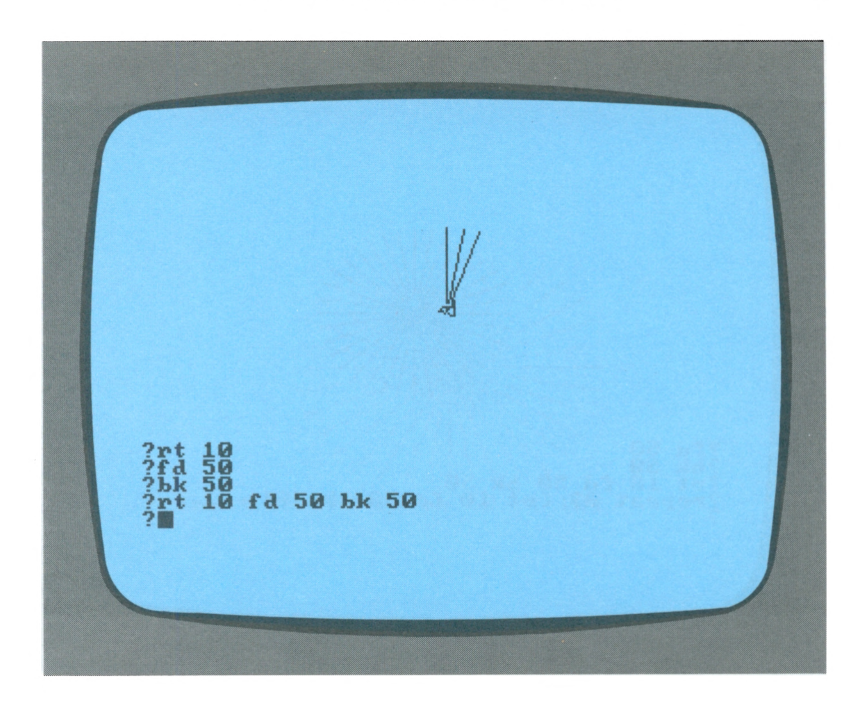
Löschen wir jetzt mit cs und ct den Bildschirm und machen eine neue Zeichnung. Die Schildkröte befindet sich in der Schirmmitte und zeigt nach oben. Wir wollen einen Strich nach oben machen und mit bk zum Ausgangspunkt zurückkehren.

```
cs
fd 50
bk 50
```



Drehen wir die Schildkröte um 10 Grad nach rechts und erzeugen den gleichen Strich.

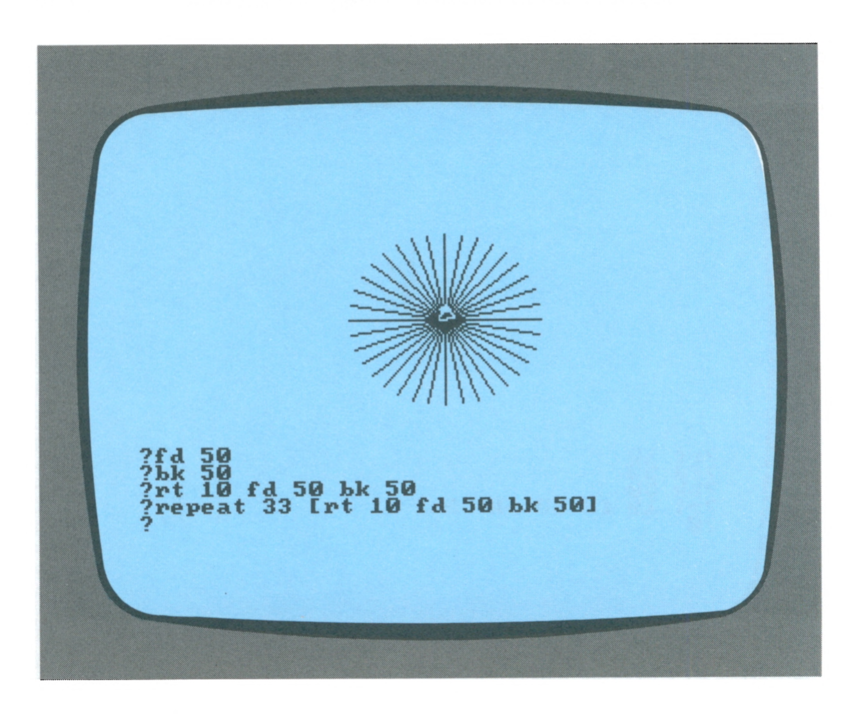
```
rt 10
fd 50
bk 50
```



Erzeugen wir noch einen dritten Strich. Bisher haben wir Befehle untereinander geschrieben. In Logo lassen sich aber Befehle ebenso nebeneinander schreiben.

```
rt 10  fd 50  bk 50
```

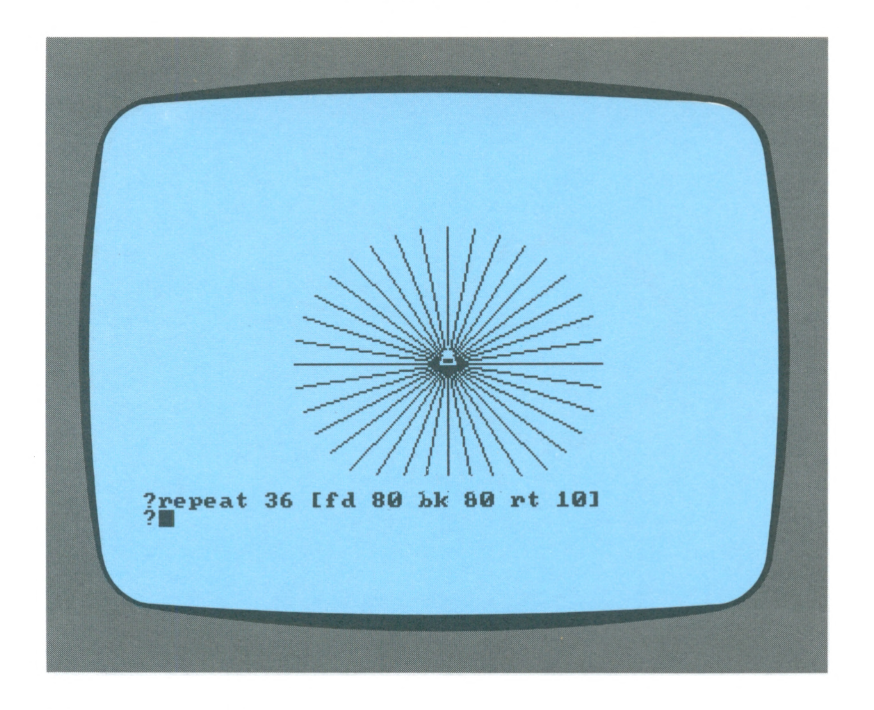
Erinnern wir uns daran, daß wir mit Leerzeichen jedes Teil einer Eingabezeile vom anderen abtrennen müssen. Also zwei Dinge nicht zusammenschreiben – oder Logo meckert. Doch dieser Fehler wird uns häufig genug widerfahren.



Wenn wir diesen Vorgang oft genug wiederholen, könnten wir ein Strahlenbündel erzeugen oder anschaulicher eine Sonne. Das ist natürlich Arbeit. Doch leichter haben wir es mit dem Wiederholungsbefehl repeat.

```
repeat 33 [rt 10 fd 50 bk 50]
```

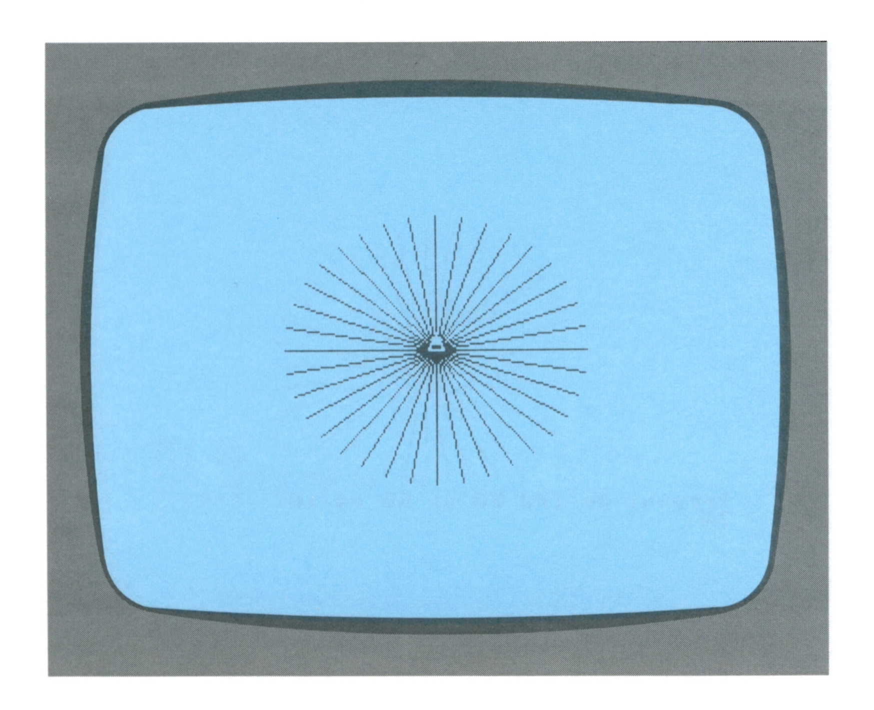
Der repeat-Befehl hat zwei Eingaben. Zuerst ist es die Anzahl der gewünschten Wiederholungen und dann sind es in eckigen Klammern alle Befehle, die entsprechend oft wiederholt ausgeführt werden sollen.



Löschen wir den Schirm mit cs und ct und erzeugen eine Sonne, deren Strahlen 80 Einheiten lang sind.

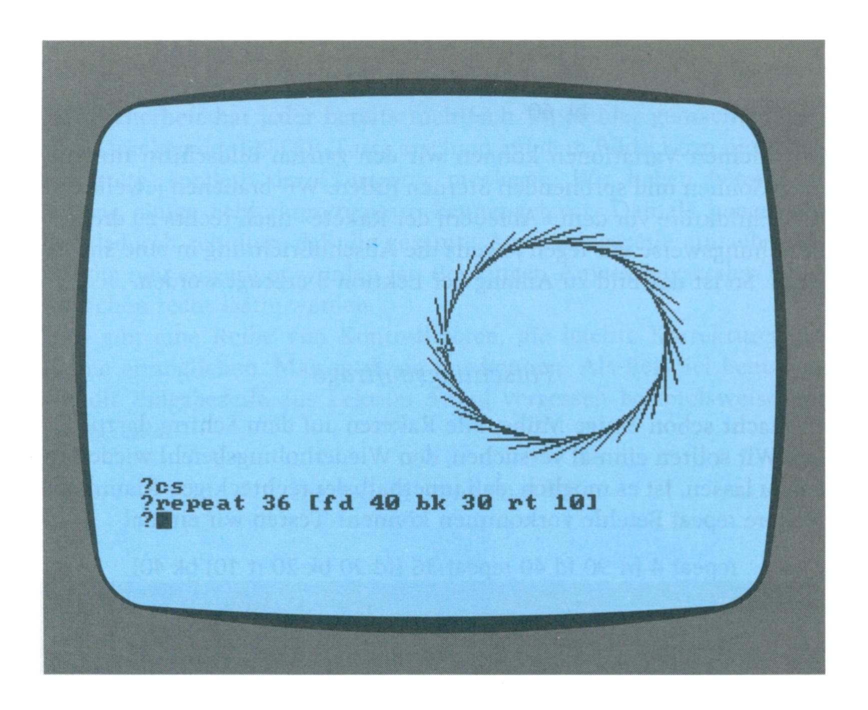
```
repeat 36 [fd 90 bk 90 rt 10]
```

Am unteren Teil des Schirms werden Striche der Zeichnung wegen des Textes nicht abgebildet.



Unterdrücken wir den Textteil mit fs. fs ist die Abkürzung von fullscreen und bedeutet «ganzer Schirm». Doch jetzt können wir nicht mehr die eingetippten Befehle kontrollieren und mitlesen.

Wollen wir die Befehle lesen, müssen wir nur den alten Zustand wiederherstellen. Der zweigeteilte Schirm wird mit dem Befehl ss erzeugt. ss ist die Abkürzung von *splitscreen* und bedeutet «geteilter Schirm».



Beim Experimentieren ist durch einen Tippfehler das Feuerrad entstanden. Statt die gleiche Anzahl Schritte wie beim Vorwärtsbefehl zurückzugehen, wurde eine kleinere Anzahl Schritte zurückgelaufen.

```
repeat 36 [fd 40 bk 30 rt 10]
```

Nachfolgend wollen wir ein kleines Silvesterfeuerwerk erzeugen. Die erste Rakete ist gezündet und explodiert. Mit bk 60 ist die Schildkröte wieder im Ausgangspunkt.

```
cs
lt 45 fd 60
repeat 36 [fd 40 bk 40 rt 10]
bk 60
```

Wir drehen die Schildkröte und feuern eine weitere Rakete ab.

```
rt 80 fd 60
repeat 36 [fd 30 bk 30 rt 10]
bk 60
```

Mit kleinen Variationen können wir den ganzen Bildschirm mit solchen Sonnen und sprühenden Sternen füllen. Wir brauchen jeweils nur die Schildkröte vor dem «Abfeuern der Rakete» nach rechts zu drehen, beziehungsweise wir legen jeweils die Abschußrichtung in eine andere Bahn. So ist das Bild zu Anfang der Lektion 3 erzeugt worden.

Forschungsaufträge

Es macht schon einige Mühe, alle Raketen auf dem Schirm darzustellen. Wir sollten einmal versuchen, den Wiederholungsbefehl wiederholen zu lassen. Ist es möglich, daß innerhalb der rechteckigen Klammern weitere repeat-Befehle vorkommen können? Testen wir einmal

```
repeat 4 [rt 90 fd 40 repeat 36 [fd 20 bk 20 rt 10] bk 40]
```

Versuchen wir dann, ob es uns gelingt, die vielen Raketen durch eine einzige Anweisungszeile zu erzeugen.

Neue Logo-Vokabeln

```
fs
ss
repeat
```

Kurzform der Lektion

Lektion 3 beschreibt die Möglichkeit, den Bildschirm zweigeteilt im Splitmodus zu benutzen oder den ganzen Schirm für Grafik einzusetzen. Im Splitmodus werden die letzten fünf Eingaben mitprotokolliert. Man kann jederzeit von einem Modus auf den anderen umschalten.

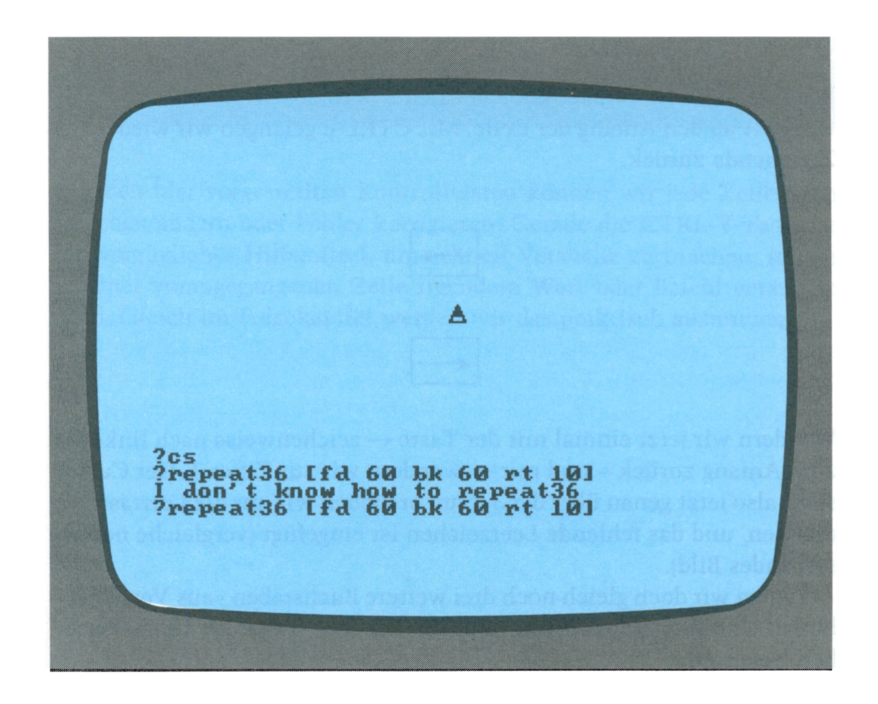
Wirkungsvoll kann der repeat-Befehl sein, der alles, was in seinen eckigen Klammern steht, beliebig oft wiederholt. Die Anzahl der Wiederholungen wird durch eine Zahl vorgegeben. repeat-Befehle können auch ineinander verschachtelt werden, wie es in den Forschungsaufträgen zu sehen ist.

Lektion 4: Tippfehler stören uns nicht

Mit Sicherheit hat jeder bereits mehrfach Tippfehler gemacht. Nach dem Drücken der ENTER-Taste erschien auf dem Bildschirm nicht das Erwartete, sondern der Computer meckerte. Wir haben bereits zu Anfang einige Fehlerkommentare kennengelernt. Damals haben wir uns dadurch geholfen, daß die gesamte Anweisungszeile einfach vollständig neu eingetippt worden ist. Bei langen Anweisungszeilen kann das schon recht lästig werden.

Es gibt eine Reihe von Kontrolltasten, die leichte Korrekturen an Zeilen ermöglichen. Man muß sie nur kennen. Als Beispiel benutzen wir die Eingabezeile aus Lektion 3 und vergessen beispielsweise ein Leerzeichen:

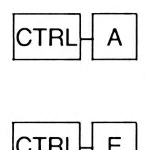
repeat36 [fd 60 bk 60 rt 10]



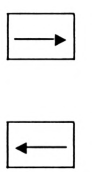
Nach dem Drücken der ENTER-Taste entsteht keine Sonne, sondern ein Fehlerkommentar. Statt die Zeile neu einzugeben, drücken wir CTRL-Y. Die eingetippte letzte Zeile erscheint erneut.



Das CTRL-Y ist schon prima. Aus einem Wiederholspeicher, in dem die letzte Eingabezeile abgelegt worden ist, wird die Zeile mit CTRL-Y erneut aufgerufen. CTRL-Y und anschließendes ENTER machen das Ganze noch einmal.

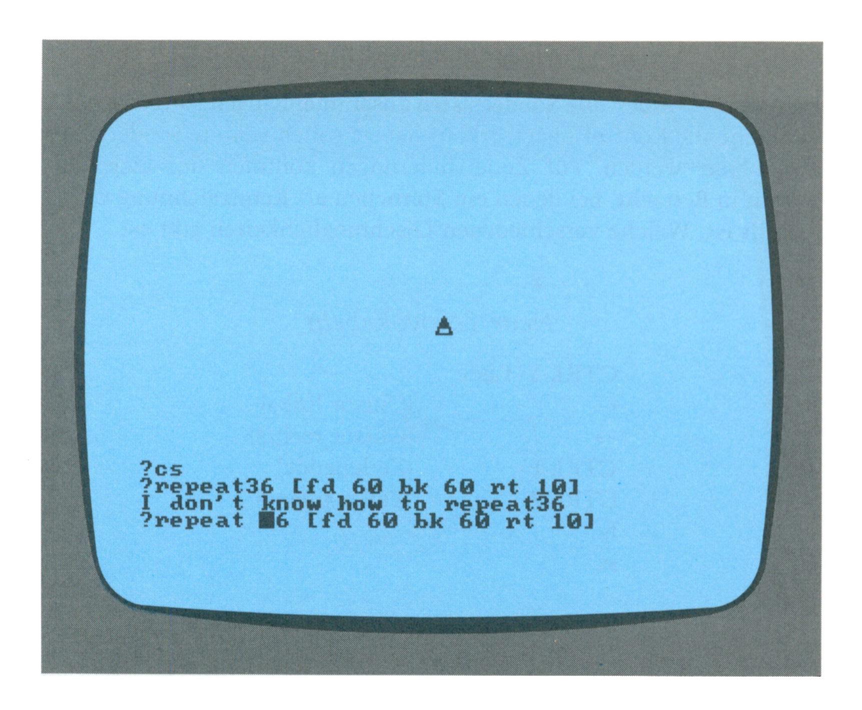


Der Cursor steht nun am Ende der Zeile. Springen wir jetzt einmal mit CTRL-A an den Anfang der Zeile. Mit CTRL-E gelangen wir wieder ans Zeilenende zurück.



Wandern wir jetzt einmal mit der Taste ← zeichenweise nach links bis zum Anfang zurück – und mit → wandern wir zur Ziffer 3. Der Cursor steht also jetzt genau über der 3. Nun brauchen wir nur die Leertaste zu drücken, und das fehlende Leerzeichen ist eingefügt (vergleiche nebenstehendes Bild).

Tippen wir doch gleich noch drei weitere Buchstaben «aus Versehen» hinzu. Und mit dreimaligem Drücken der DEL-Taste ist dieses Versehen beseitigt.



Mit den hier vorgestellten Kontrolltasten können wir jede Zeile nach Belieben ändern oder Fehler korrigieren. Gerade die CTRL-Y-Taste ist ein vorzügliches Hilfsmittel, um schnell Versuche zu machen, indem in einer vorangegangenen Zeile irgendein Wert oder Befehl verändert wird. Gleich im Folgekapitel werden wir das praktisch ausnutzen.

Forschungsauftrag

Im Anhang werden alle Sondertasten zusammenhängend angeführt. Da wir nicht alle Kontrolltasten erklärt haben, sollen weitere Sondertasten ausprobiert werden. Für Zeilenänderungen kommen nur diejenigen Tasten in Betracht, bei denen ein Sternchen als Kennzeichnung vorangestellt ist. Welche verschiedenen Löschmöglichkeiten gibt es?

Neue Logovokabeln

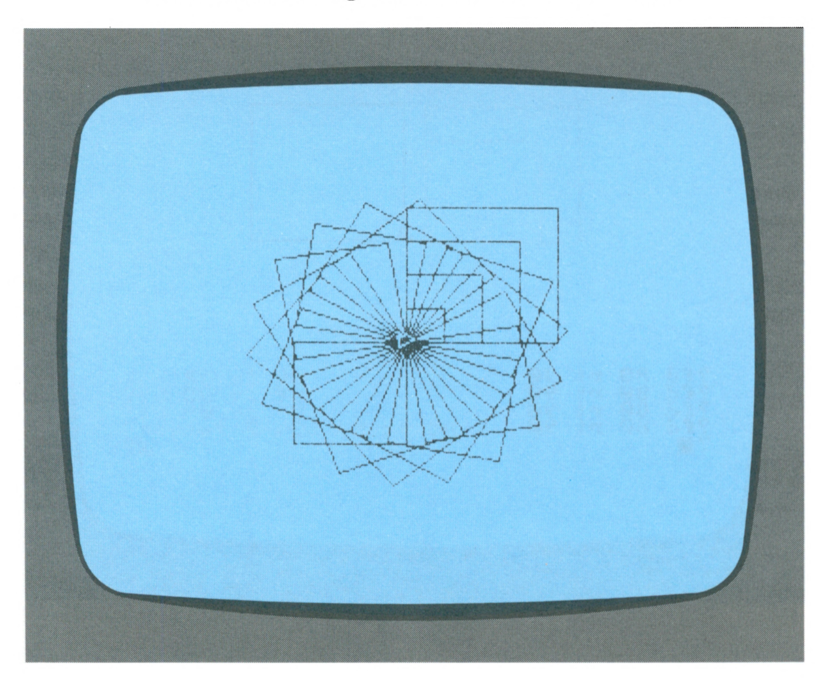
CTRL-Y-Taste	
←	(Cursor links)
→	(Cursor rechts)
CTRL-E	(Zeilenende)
CTRL-A	(Zeilenanfang)
fs	
ss	

Kurzform der Lektion

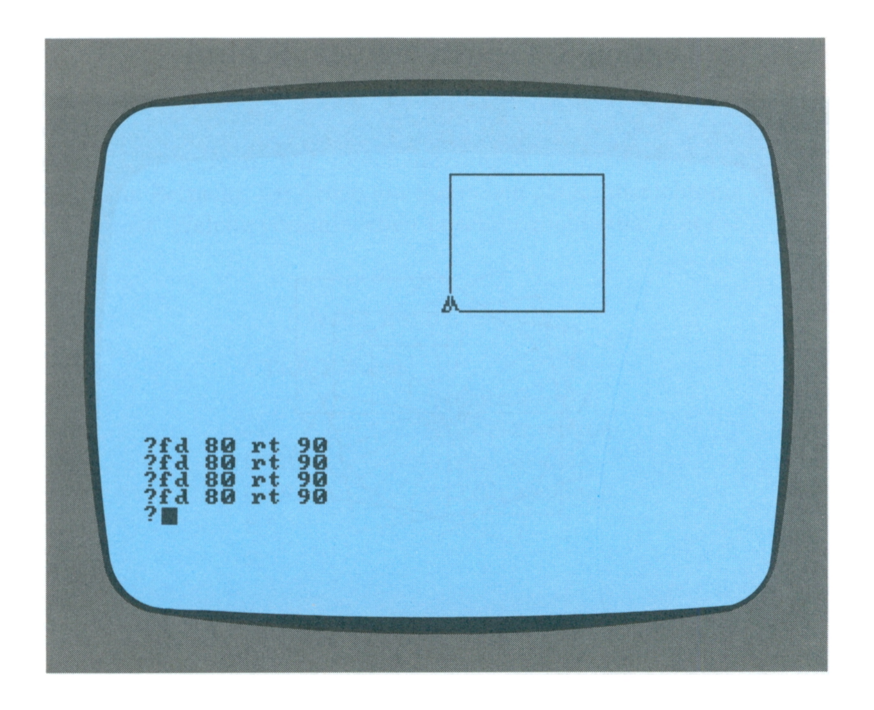
Wichtig ist die Fertigkeit, Tippfehler in Eingabezeilen zu korrigieren. Jedes Zeichen einer Zeile läßt sich direkt ansteuern. Die oben angeführten sechs Kontrolltasten sind ausreichend für alle Änderungsvorhaben. Beliebige Zeichen können eingefügt und gelöscht werden. Von besonderem Interesse ist CTRL-Y. Hiermit wird die letzte Anweisungszeile aufgerufen. Nach Drücken von ENTER wird sie erneut vom Computer ausgeführt. Bei Tests oder Fehlern können bequem Veränderungen oder Korrekturen erfolgen.



Lektion 5: Figuren auf dem Schirm



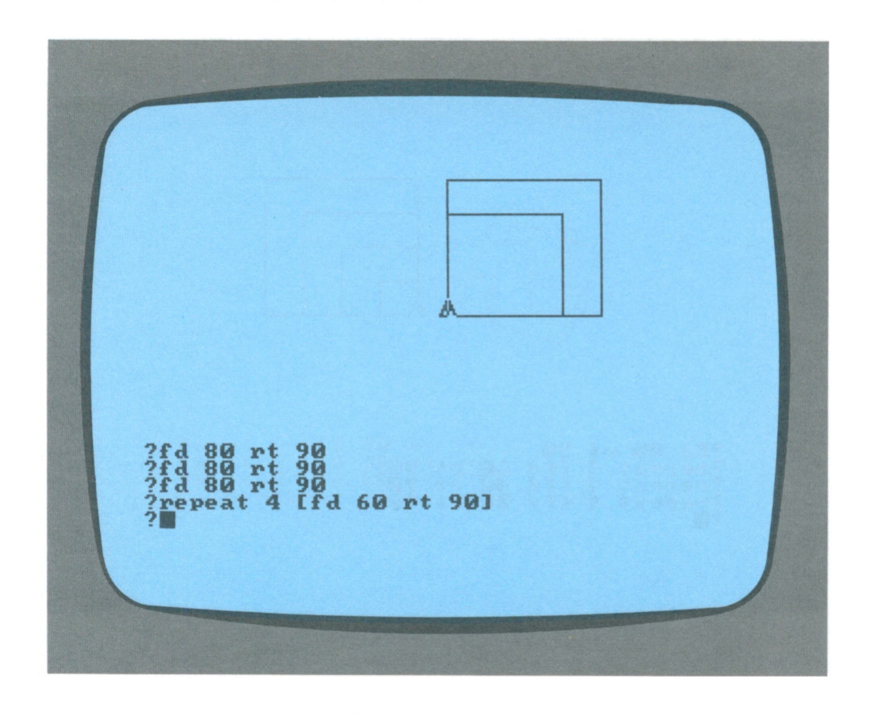
In Lektion 2 haben wir Striche immer von der Bildschirmmitte ausgehen lassen. Jetzt wollen wir noch weiter mit Strichen experimentieren. Fortgefahren wird dort, wo der letzte Strich endete.



Zeichnen wir einmal ein Quadrat mit der Seitenlänge 60.

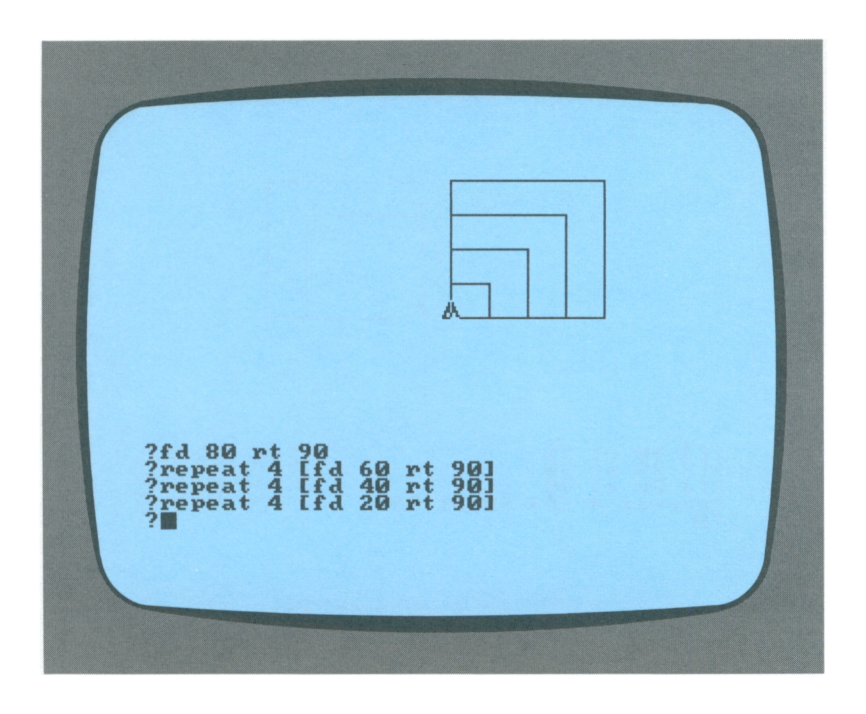
```
cs ct
fd 80 rt 90
CTRL-Y-Taste
CTRL-Y-Taste
CTRL-Y-Taste
```

Auch nach dem CTRL-Y müssen wir jedesmal die ENTER-Taste drükken. Das CTRL-Y wird auf dem Bildschirm nicht mitprotokolliert, sondern die erneut aufgerufene und ausgeführte Anweisungszeile.

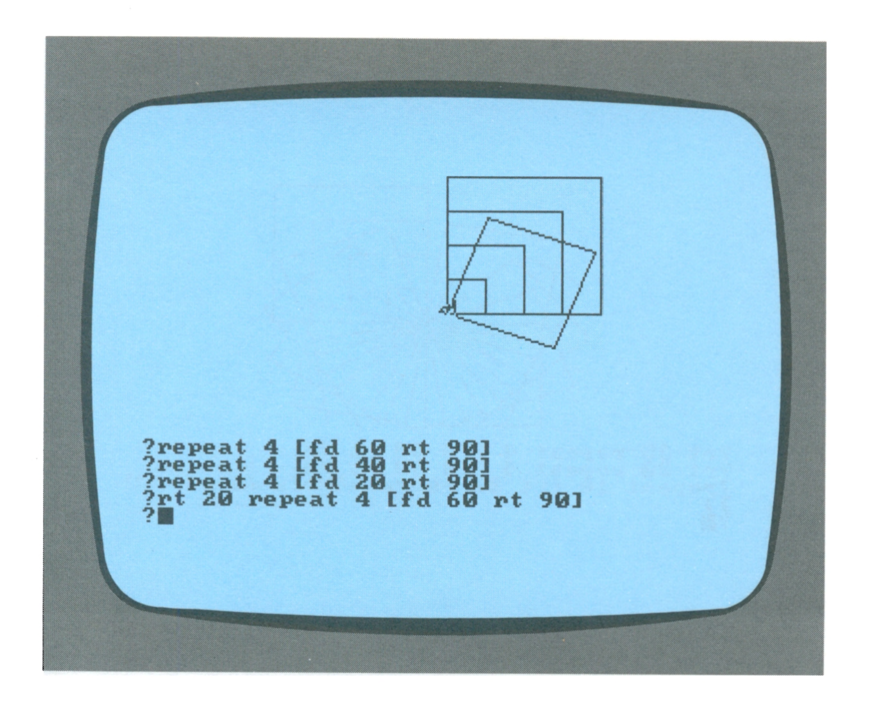


Als nächstes zeichnen wir in das große Quadrat drei kleinere ein. Benutzen wir jetzt aber den repeat-Befehl.

```
repeat 4 [fd 60 rt 90]
```



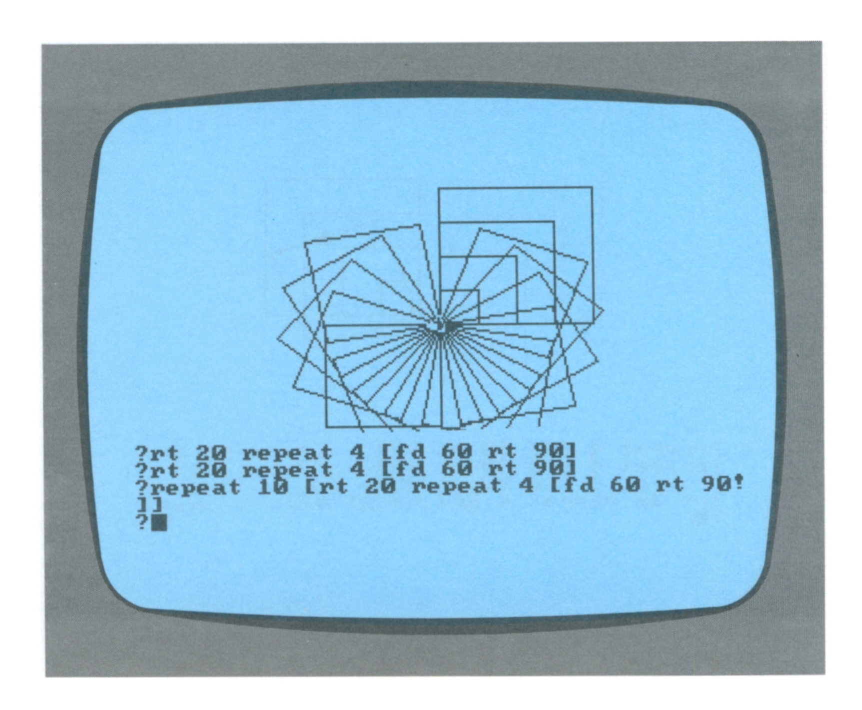
Die nächsten zwei Quadrate sollen die Seitenlänge 40 und 20 haben. Benutzen wir hierzu die nützliche CTRL-Y-Taste. Doch nach jedem CTRL-Y muß noch die Zahl für den Vorwärtsbefehl fd geändert werden. Der Cursor steht am Ende der Anweisungszeile. Wandern wir also so lange in der Zeile nach links zurück (mit ←), bis der Cursor genau über der Ziffer 0 steht. Jetzt muß man nur noch die Taste DEL drücken und die Ziffer 4 eintippen. Springen wir an das Zeilenende zurück und drücken ENTER. In gleicher Weise verfahren wir beim Zeichen des kleinsten Quadrats.



Jetzt soll das Quadrat mit der Seitenlänge 60 gedreht werden. Erinnern wir uns an den kleinen Trick bei der Sonne in Lektion 2. Drehen wir die Schildkröte um 20 Grad, aber – halt! – diesmal nicht gleich das ENTER anschließen.

rt 20 (kein ENTER)
Leertaste drücken
CTRL-Y
20 in 60 umändern
ans Zeilenende springen und ENTER

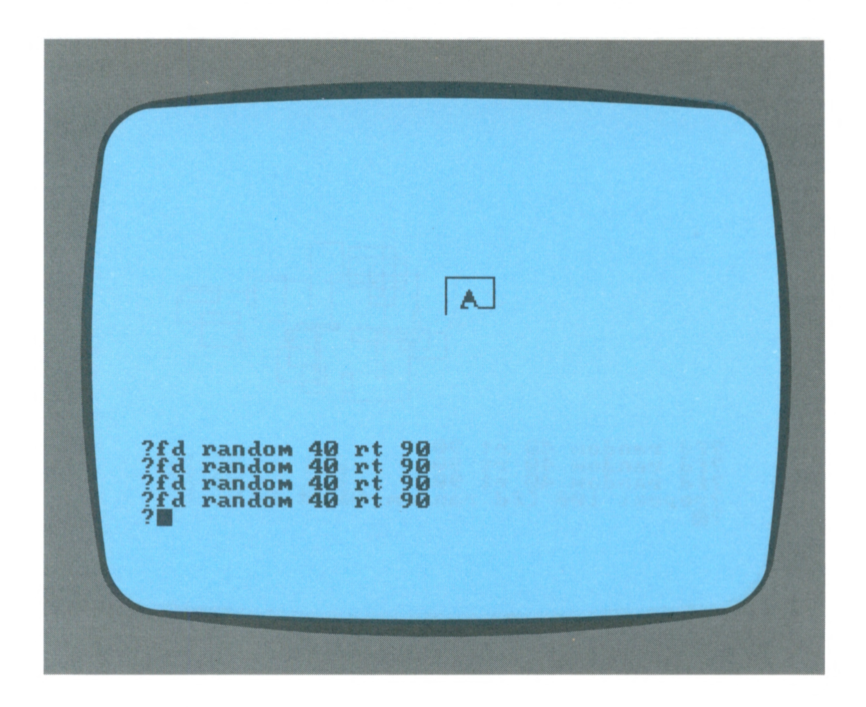
Das Ändern einer Eingabezeile in Verbindung mit CTRL-Y erspart uns viel Tipperei beim Experimentieren.



Wiederholen wir jetzt oft genug mit CTRL-Y die Anweisungszeile, so entsteht dieses Bild. Statt zehnmal CTRL-Y und ENTER zu drücken, hätten wir auch in der Anweisungszeile ein zusätzliches repeat 10 hinzufügen können. Erinnert sei an die Befehle fs und ss (vgl. Lektion 3).

repeat 10 (eintippen)
Leerzeichen als Trenner
CTRL-Y
ans Zeilenende und ENTER

Wir wollen noch ein wenig weiter mit dem Strich als Grundelement experimentieren. Jetzt soll die jeweilige Schrittlänge ständig geändert werden. Hierzu wollen wir die Schrittlängen durch einen Zufallsgenerator erzeugen lassen. Zufallszahlen werden mit random (Zufall) erzeugt. Löschen wir den Schirm, und wiederholen wir mehrfach mit CTRL-Y die Anweisung: fd random 20.

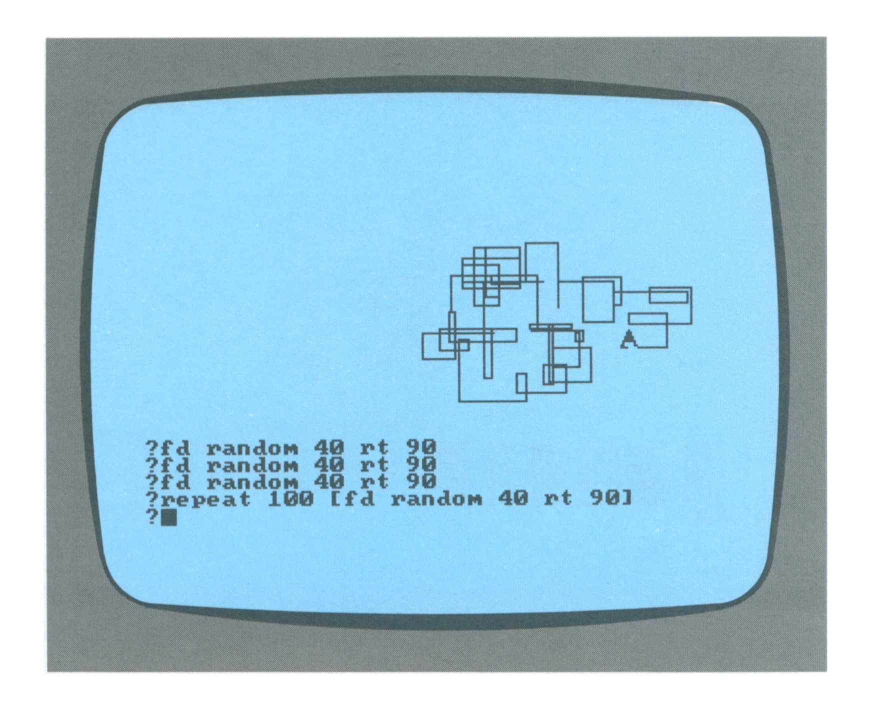


Richtig beobachtet! Es entstehen Striche unterschiedlicher Länge. Die maximale Länge wird durch die Zahl 20 vorgegeben. random 20 erzeugt Zufallszahlen von 0 bis 19. Würden wir eine größere Zahl wählen, beispielsweise 50, entstünden Zufallszahlen im Bereich von 0 bis 49. Die maximal erzeugte Zufallszahl ist also jeweils um eins kleiner als die angegebene Obergrenze.

Löschen wir erneut den Schirm und testen die folgende Anweisung:

```
fd random 40 rt 90
```

Wiederholen wir die Anweisung noch dreimal mittels CTRL-Y. Das Quadrat schließt sich nicht mehr. Die Seitenlänge hat sich ja geändert, obwohl sich die Schildkröte einmal um sich selbst gedreht hat und wieder genau die Richtung nach oben einnimmt.

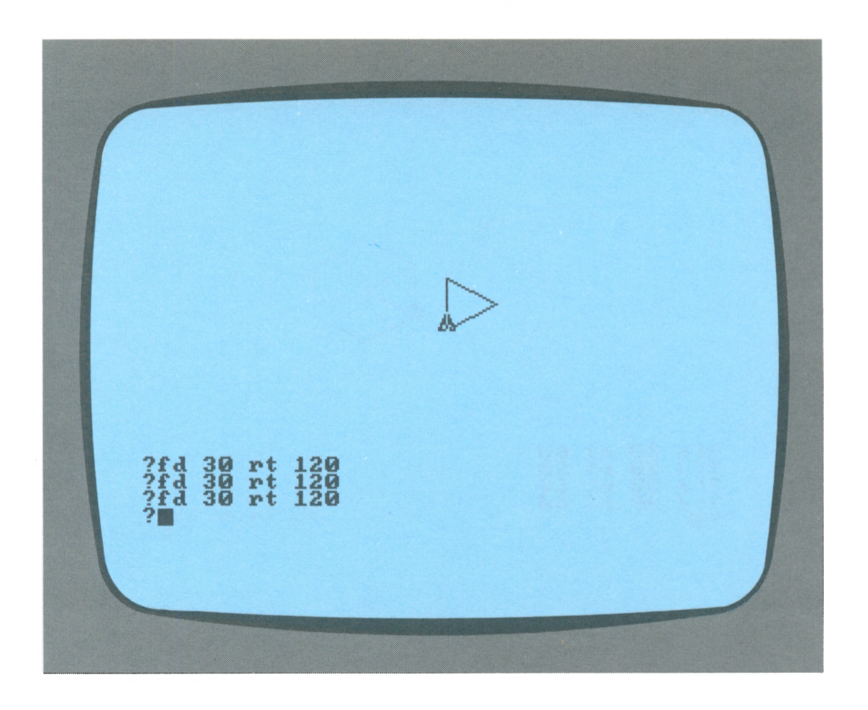


Statt ständig CTRL-Y zu hämmern, benutzen wir besser den repeat-Befehl. Lassen wir den sich drehenden, aber unterschiedlich langen Strich 100mal von der Schildkröte zeichnen.

repeat 100 [fd random 40 rt 90]

Die Zeichnungen wirken doch interessant, und wer Lust hat, soll, bevor er weiterliest, damit noch experimentieren.

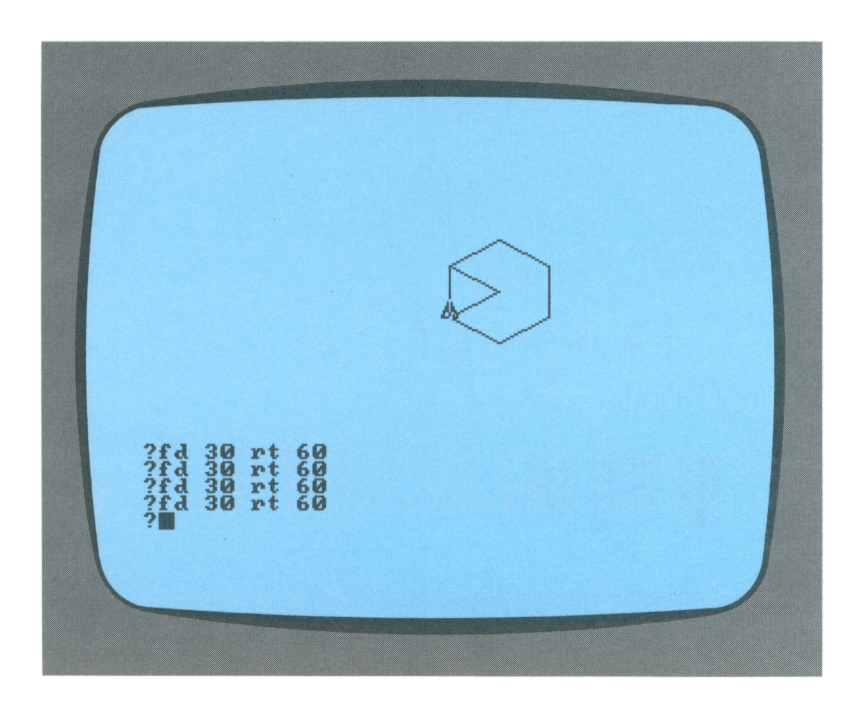
Bisher haben wir in den Wiederholungen die Länge des Vorwärtsbefehls geändert und die Gradzahl für die Rechtsdrehung immer konstant mit 90 beibehalten. Nachfolgend experimentieren wir mit Figuren, die durch Veränderung der Zahleneingabe beim rt-Befehl entstehen können.



Testen wir einmal die nachfolgende Anweisung und wiederholen sie dreimal.

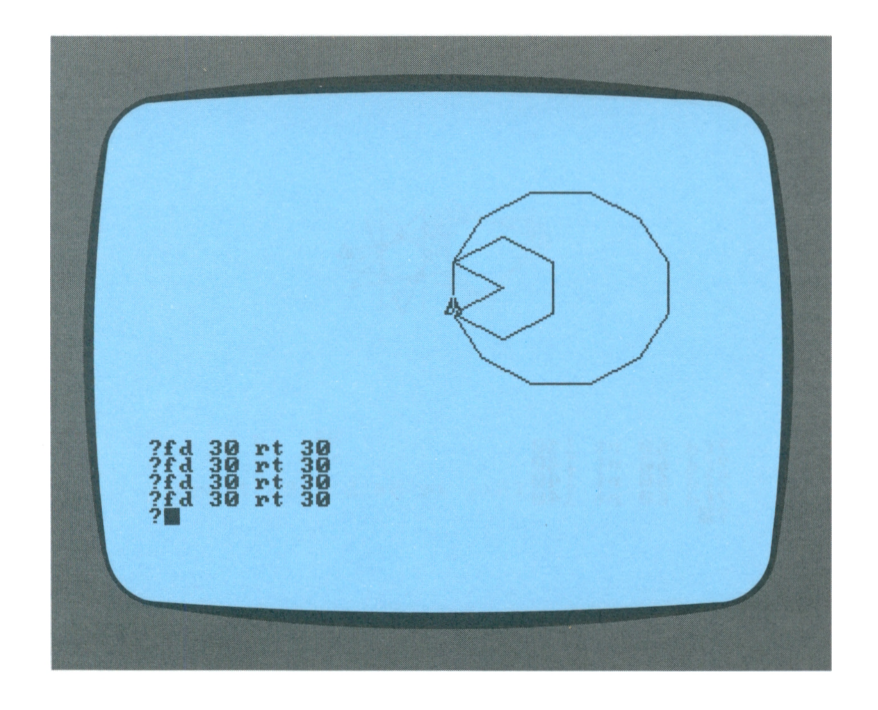
```
fd 30 rt 120
```

Es entsteht ein gleichseitiges Dreieck.

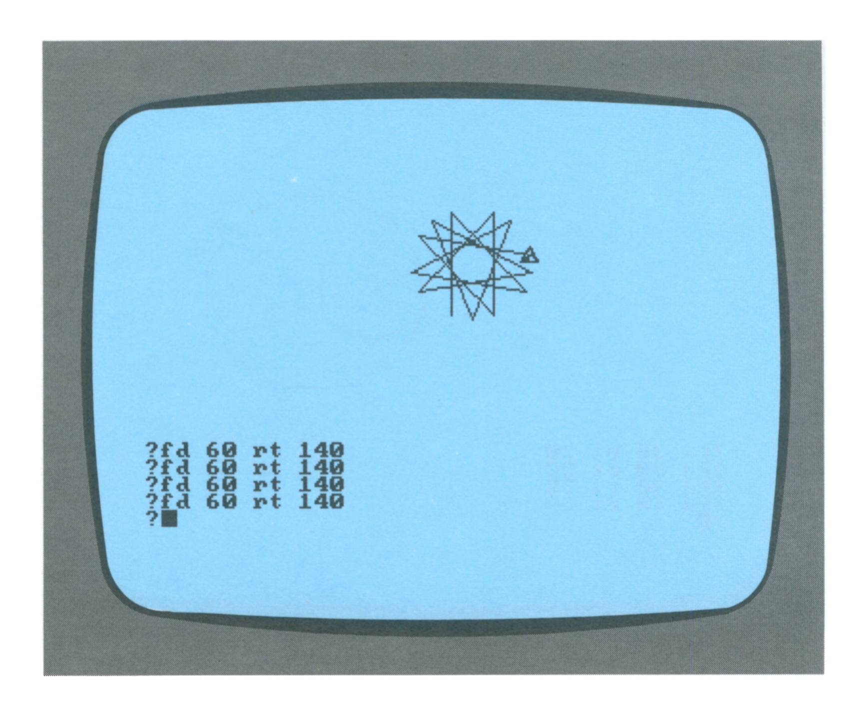


Testen wir die nächste Anweisung und wiederholen sie mehrfach mit CTRL-Y, bis sich die Figur schließt.

fd 30 rt 60



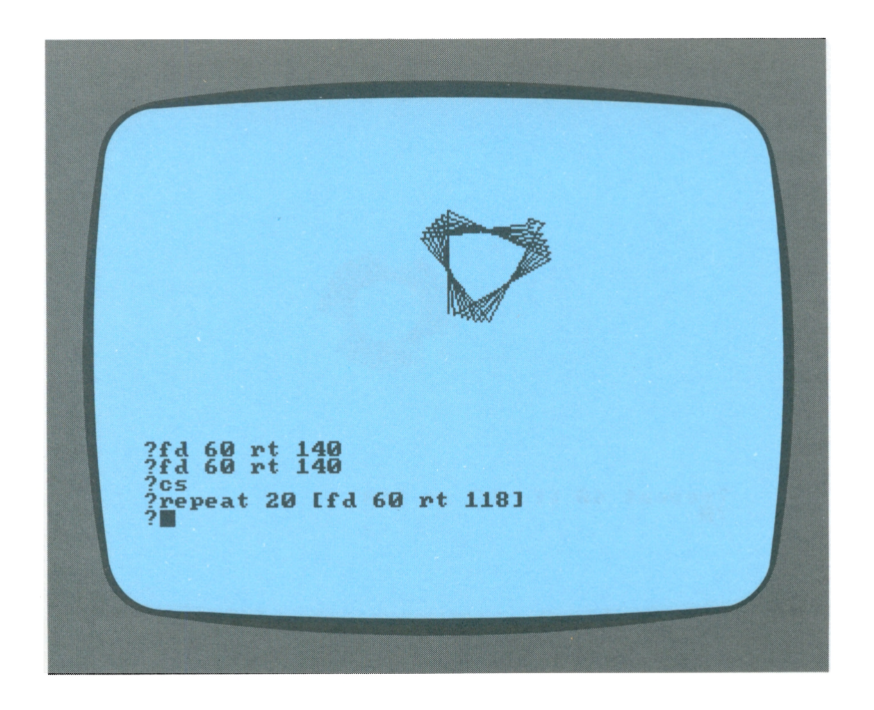
Wenn wir das mit 30 Grad wiederholen, entsteht ein Zwölfeck.



Mit den Drehwinkeln sind wir bisher kleiner geworden. Testen wir jetzt einmal größer werdende Gradzahlen.

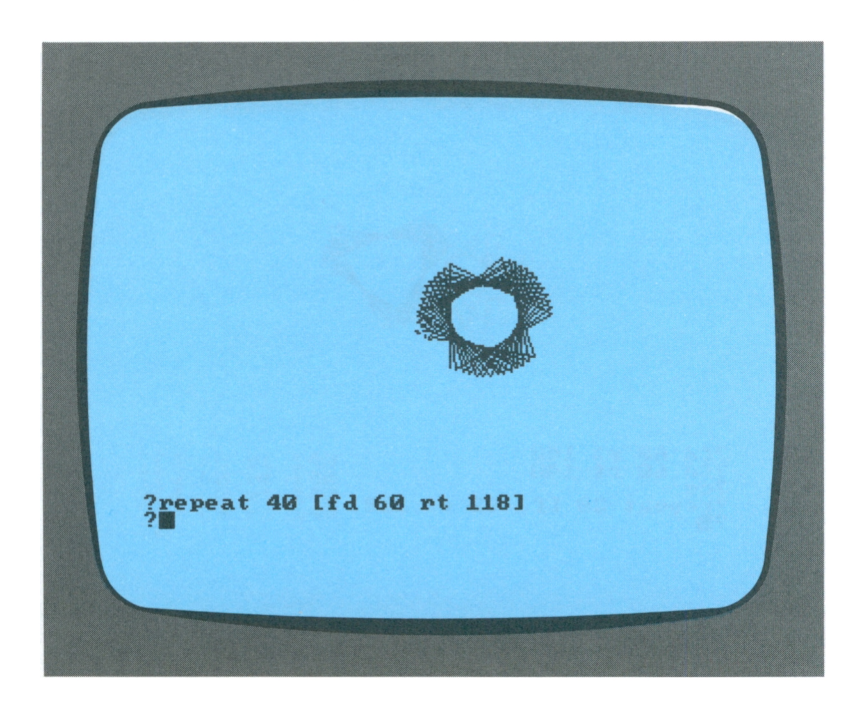
```
cs
fd 60 rt 140
mehrfach CTRL-Y
```

Es entsteht eine Sternfigur.

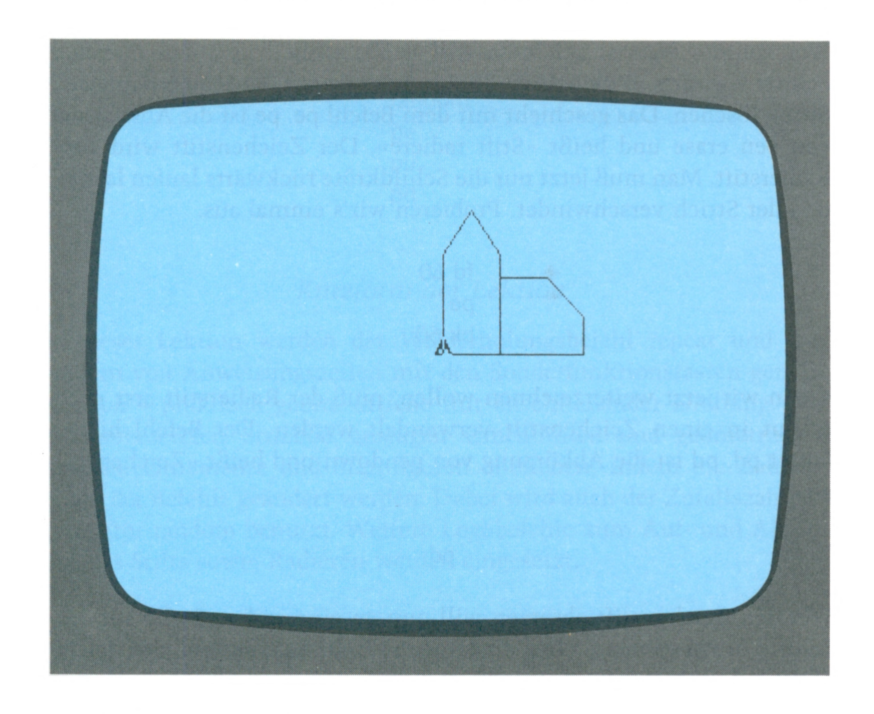


Wiederholen wir das gleiche einmal mit einer Gradzahl von vielleicht 118 und schauen uns das Ergebnis an.

```
cs
repeat 20 [fd 60 rt 118]
```



Ist die Anzahl der Wiederholungen groß genug, schließt sich die Figur zu einem Superstern.



Neben diesen mehr geometrischen Experimenten lassen sich natürlich viele Figuren und Umrandungslinien zeichnen.

```
fd 70 rt 30
fd 30 rt 120
fd 30 rt 30
fd 60 bk 30 rt 90

fd 46 rt 90
fd 25 rt 50
fd 30 rt 40
fd 23 rt 90
rt 90 fd 75
rt 90
```

Wer weitere Figuren zeichnen will, stellt schnell fest, daß mancher Schritt vorwärts in die falsche Richtung gegangen ist. Man möchte den Strich löschen. Das geschieht mit dem Befehl pe. pe ist die Abkürzung von pen erase und heißt «Stift radiere». Der Zeichenstift wird zum Radierstift. Man muß jetzt nur die Schildkröte rückwärts laufen lassen, und der Strich verschwindet. Probieren wir's einmal aus.

```
fd 60
pe
bk 60
```

Wenn wir jetzt weiterzeichnen wollen, muß der Radierstift erst noch erneut in einen Zeichenstift verwandelt werden. Der Befehl hierzu lautet pd. pd ist die Abkürzung von pendown und heißt «Zeichenstift ab».

```
pd
```

Wer den Zeichenstift absetzen will, um an eine andere Stelle auf den Schirm hinzuwandern, ohne eine Spur zu hinterlassen, muß den Befehl «Stift auf» geben (penup, abgekürzt pu):

```
pu
```

Durch geschickten Einsatz von pu und pd lassen sich beliebige Zeichnungen erstellen. Zum Verdeutlichen soll jeder einmal die Anfangsbuchstaben seines Namens in Blockbuchstaben von der Schildkröte zeichnen lassen.

Forschungsaufträge

Die Schildkröte soll eine geschlossene Fläche durch Striche erzeugen. Hinweis: Auf- und Abstriche, die jeweils genau einen Schritt nebeneinanderliegen. Dann sollen Figuren herausradiert werden.

Entsteht Kritzelei oder vielleicht sogar Computerkunst, wenn wir einen Strich hundertfach wiederholen, wobei Schrittweite und Drehwinkel jeweils Zufallszahlen mittels random als Werte erhalten? (Ausprobieren!)

Neue Logovokabeln

random
pe
pu
pd

Kurzform der Lektion

In dieser Lektion werden der Wiederholungsbefehl repeat und das Ändern von Anweisungszeilen mit den Sonderfunktionstasten gemäß Lektion 4 praktisch geübt. Striche mit anschließender Drehung und neu angesetzten Strichen erzeugen Umrißlinien und geometrische Figuren. Ornamente und Sternfiguren entstehen, indem die Zahlenwerte der Befehle geändert werden. Dabei wird auch der Zufallszahlengenerator random benutzt. Weitere Logobefehle zum Auf- und Absetzen des Stifts sowie Radieren werden eingesetzt.

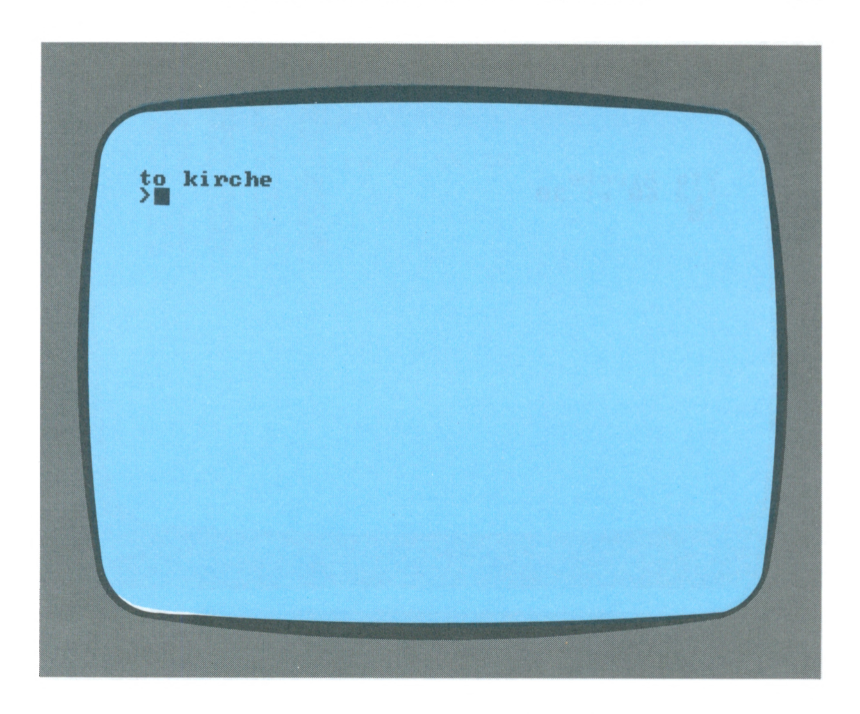
Lektion 6: So erstellt man Programme

Alle Experimente und Beispiele der vorangegangenen Lektionen haben Mühe bereitet. Gerade das Kirchenbeispiel war aufwendig. Der größte Nachteil bestand darin, daß die Beispiele nach dem Ausführen oder Abschalten des Computers verloren waren. Wir lernen nachfolgend einen Weg kennen, um die bestehenden Logobefehle mit von uns selbst definierten Befehlen zu erweitern. Solche Benutzerbefehle nennt man auch Programme. Nachfolgend fassen wir unser Kirchenbeispiel aus Lektion 5 zusammen und geben der Zusammenfassung (Programm) den Namen kirche. Anschließend muß nur noch das Wort kirche eingetippt werden, und es entsteht die Zeichnung.

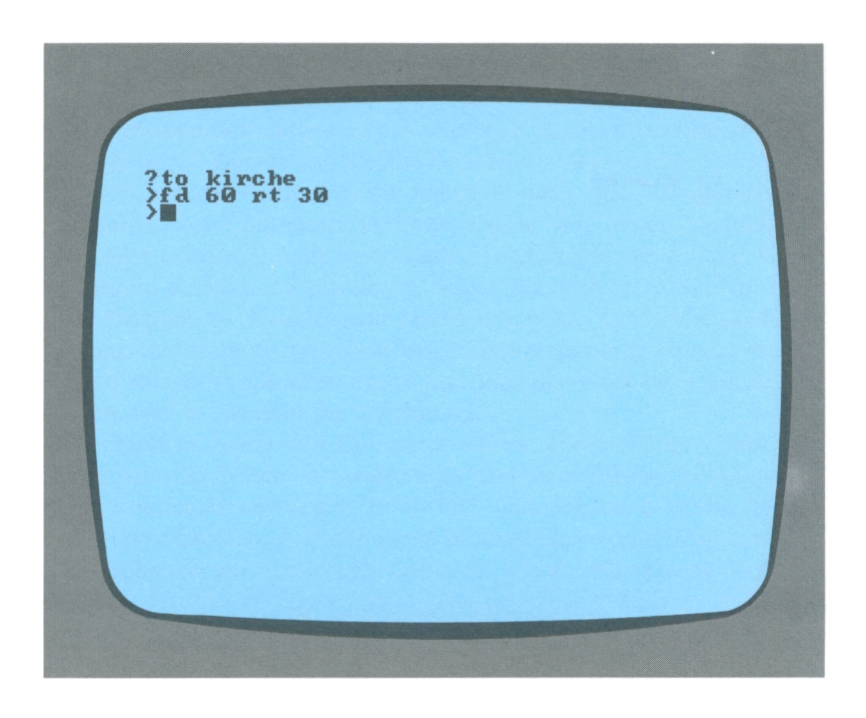
Wir müssen zuerst den Computer lehren, andere Dinge – nämlich unsere eigenen Worte – zu verstehen und auszuführen. Löschen wir zuerst mit der Anweisung ts ct ⟨ENTER⟩ den Schirm. ts ist die Abkürzung von textscreen, d. h. der ganze Bildschirm wird für Text genutzt. Mit ts schalten wir den Grafikmodus ab. Mit der Anweisung

to kirche ⟨ENTER⟩

zwingen wir den Computer auf die Schulbank. Er nimmt unsere Anweisungen zum Thema kirche entgegen. Der Befehl to veranlaßt ihn, im Wort kirche den Namen eines Programms zu sehen.



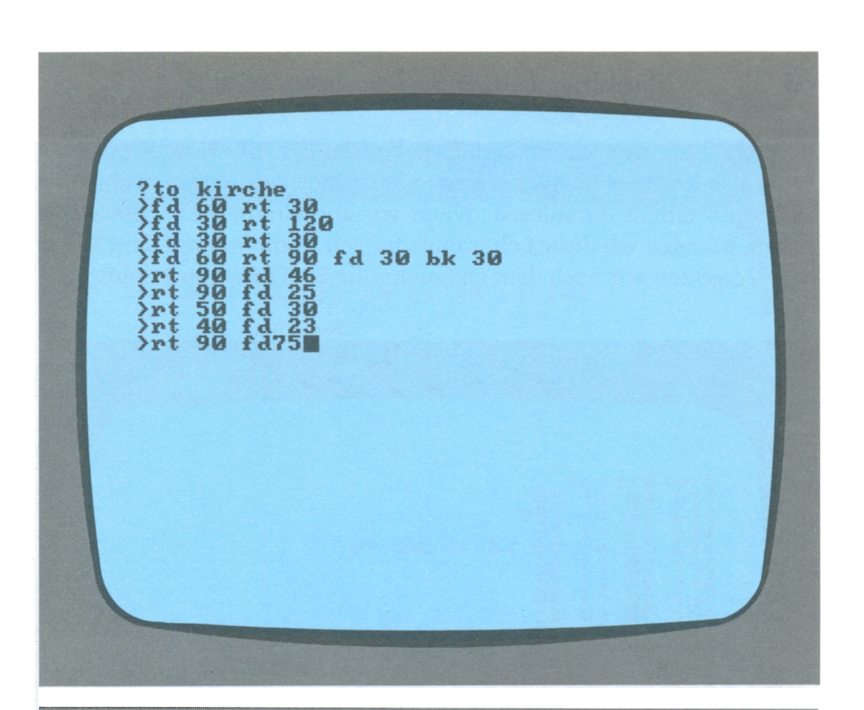
Nach dem Drücken der ENTER-Taste ändert sich der Bildschirm. to kirche steht als Kopfzeile da. Das Promptzeichen hat von «?» zu «〉» gewechselt. Nur in diesem Zustand können wir eigene Programme definieren.



Geben wir jetzt eine erste Zeile ein:

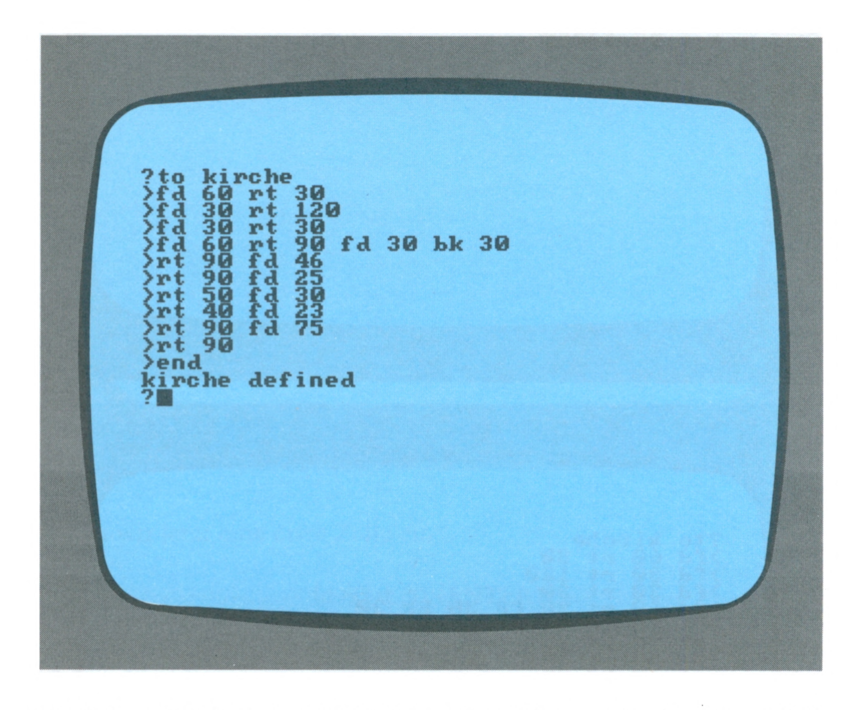
fd 60 rt 30 ⟨ENTER⟩

Nach dem Drücken der ENTER-Taste erscheint das Promptzeichen in der Folgezeile.

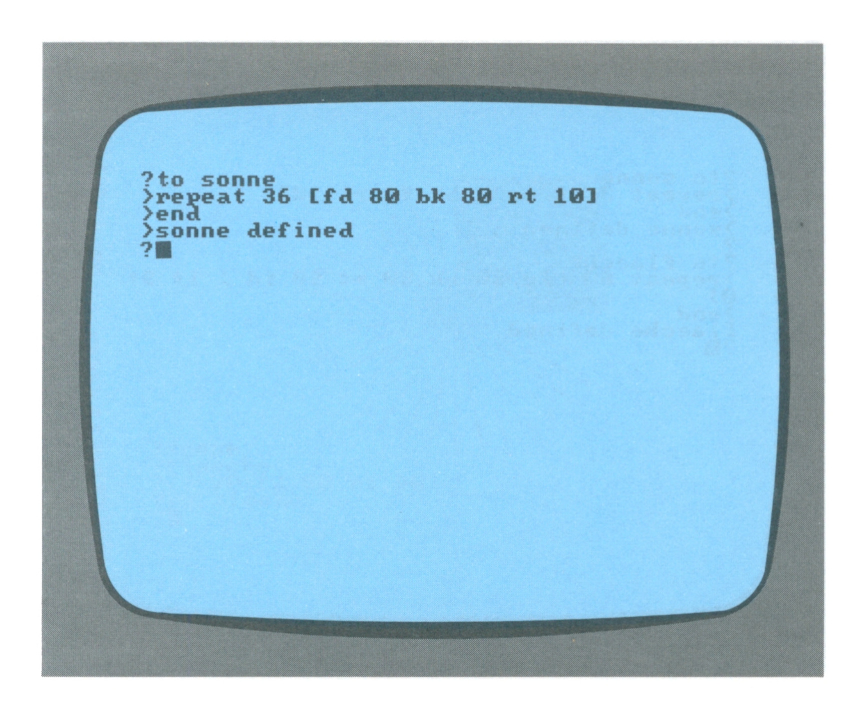



```
?to kirche
>fd 60 rt 30
>fd 30 rt 120
>fd 30 rt 30
>fd 60 rt 90 fd 30 bk 30
>rt 90 fd 46
>rt 90 fd 25
>rt 50 fd 30
>rt 40 fd 23
>rt 90 fd 75
>rt 90
>end
```

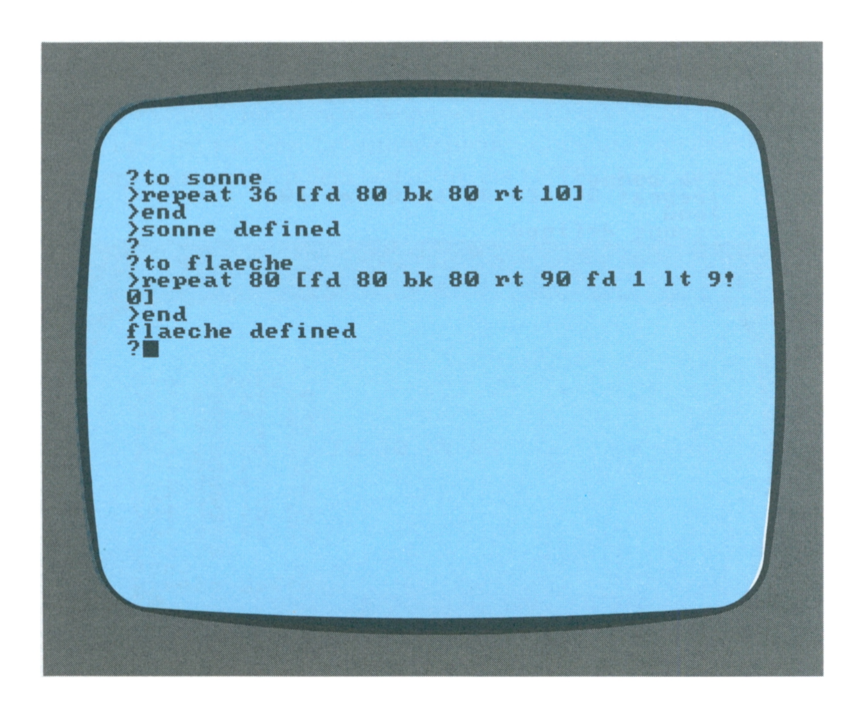
Tippfehler können wir vor dem Drücken der ENTER-Taste in gewohnter Weise korrigieren. Wer das noch einmal genau wissen will, sollte Lektion 4 erneut durchlesen. Wenn wir alle Grafikbefehle eingetippt haben, beenden wir diese Lehrstunde für den Computer mit dem Befehl end. Vergessen wir nach dem end nicht, die ENTER-Taste zu drücken!



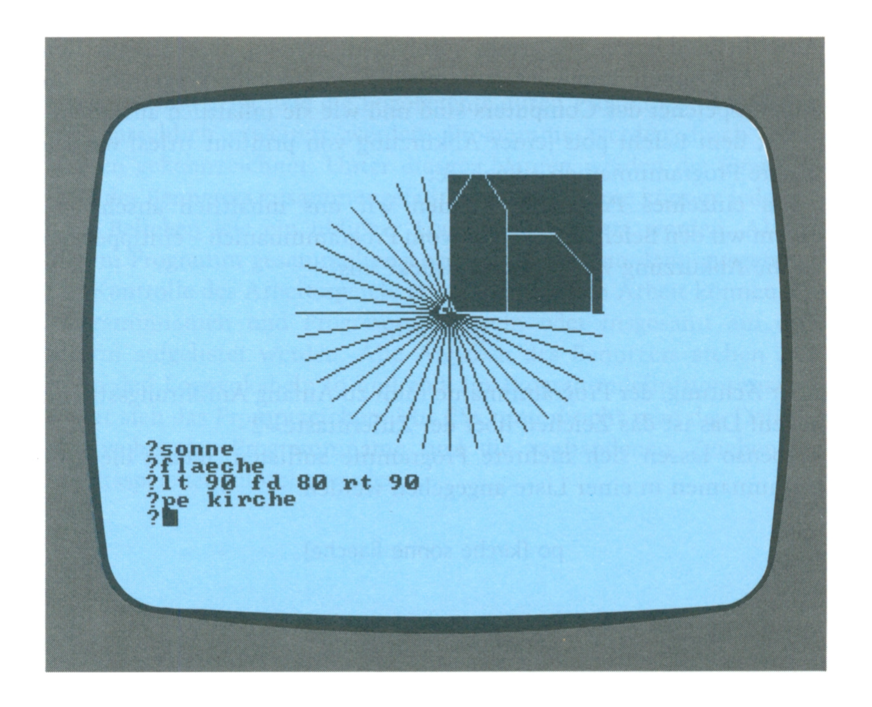
Nach dem Drücken der ENTER-Taste bestätigt uns der Computer, daß er seine Lektion gelernt hat. Gemeldet wird kirche defined. Auf deutsch heißt das «Kirche definiert». Testen wir jetzt unser erstes Programm. Löschen wir aber vorher noch den Schirm.



Erstellen wir auf gleiche Weise einmal ein Sonnenprogramm gemäß Lektion 3.



Auf gleiche Weise erstellen wir eine Fläche. Durch Auf- und Abstriche bei genügender Wiederholung ist das schnell erledigt. Sehen wir uns das Programm einmal an. Der Strich geht rauf und runter. Dann marschiert die Schildkröte einen Schritt nach rechts, wird nach oben ausgerichtet. Das Ganze wiederholt sich in diesem Fall 80mal.



Der Vorteil eigener Spracherweiterungen in Form von Programmen wird sofort klar. Geben wir einmal folgendes ein:

```
cs
sonne
flaeche
lt 90 fd 80 rt 90
pe
kirche
```

Ein schönes Ornament entsteht, wenn wir unsere Kirche sich drehen lassen

```
cs
repeat 36 [kirche rt 10]
```

Abschließend lernen wir einige Logobefehle kennen, die recht nützlich sind. Mit ihnen können wir kontrollieren, welche Programme im Arbeitsspeicher des Computers sind und wie sie inhaltlich aussehen.

Mit dem Befehl pots (einer Abkürzung von printout titles) werden unsere Programmnamen aufgelistet.

Ein einzelnes Programm können wir uns inhaltlich anschauen, indem wir den Befehl po – gefolgt vom Programmnamen – eintippen. po ist die Abkürzung von printout (drucke aus).

```
po "kirche
```

Aber Achtung, der Programmname muß zu Anfang Anführungsstriche haben! Das ist das Zeichen über der Zifferntaste «2».

Ebenso lassen sich mehrere Programme auflisten, indem die Programmnamen in einer Liste angegeben werden.

```
po [kirche sonne flaeche]
```

Forschungsauftrag

Was passiert, wenn wir während des Programmdefinierens nicht ordnungsgemäß mit end ⟨ENTER⟩ abschließen, sondern die ESC-Taste drücken? Kontrollieren wir anschließend mit pots, welche Programme im Arbeitsspeicher sind.

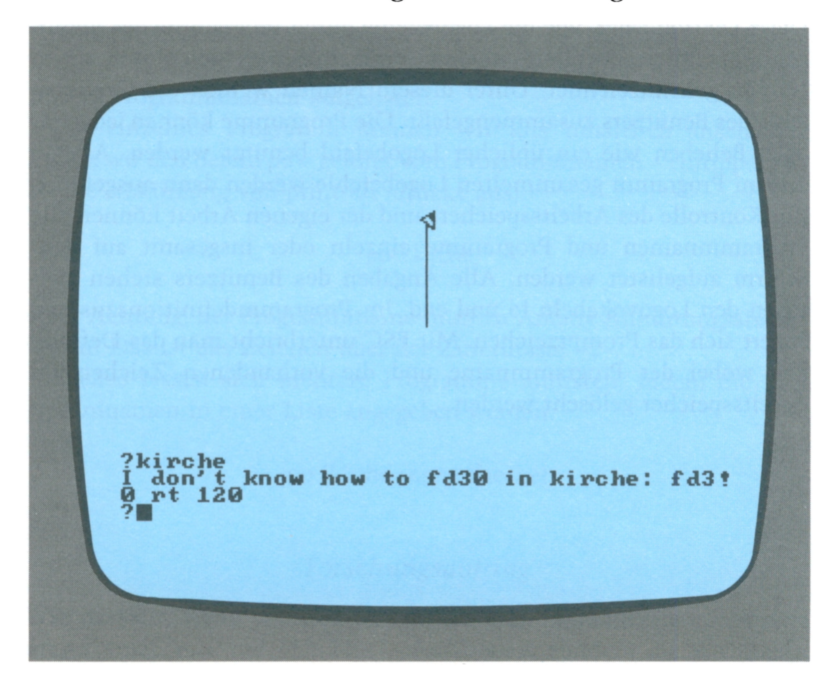
Neue Logovokabeln

to
end
pots
po
ts

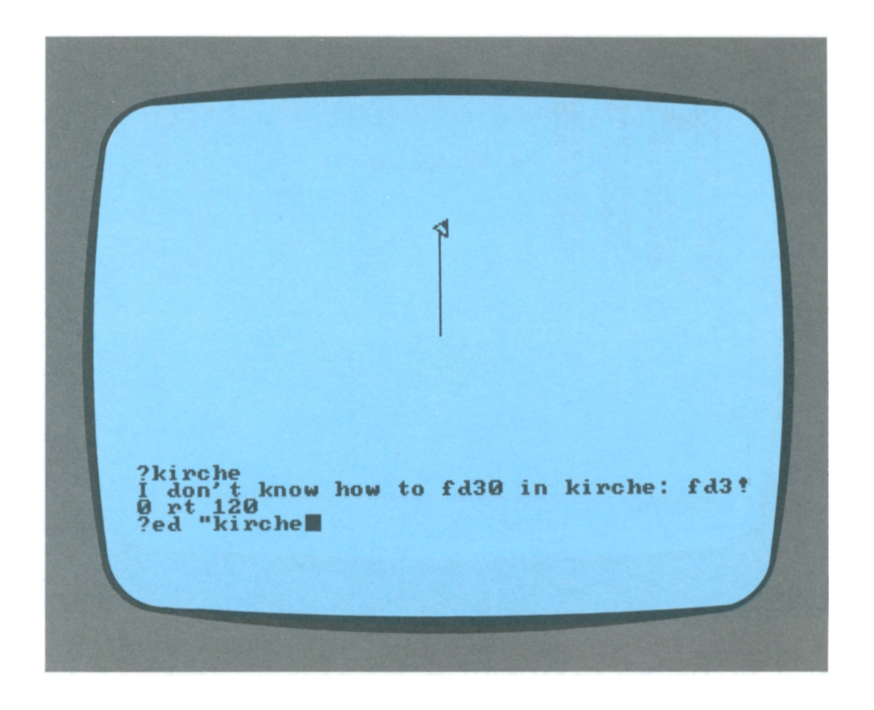
Kurzform der Lektion

Diese Lektion zeigt, wie die Logobefehle durch Programme des Benutzers sprachlich erweitert werden. Programme werden durch einen Namen gekennzeichnet. Unter diesem Namen werden die Grafikbefehle des Benutzers zusammengefaßt. Die Programme können jederzeit nach Belieben wie ein üblicher Logobefehl benutzt werden. Alle in diesem Programm gesammelten Logobefehle werden dann ausgeführt. Zur Kontrolle des Arbeitsspeichers und der eigenen Arbeit können alle Programmnamen und Programme einzeln oder insgesamt auf dem Schirm aufgelistet werden. Alle Angaben des Benutzers stehen zwischen den Logovokabeln to und end. Im Programmdefinitionszustand ändert sich das Promptzeichen. Mit ESC unterbricht man das Definieren, wobei der Programmname und die vorhandenen Zeichen im Arbeitsspeicher gelöscht werden.

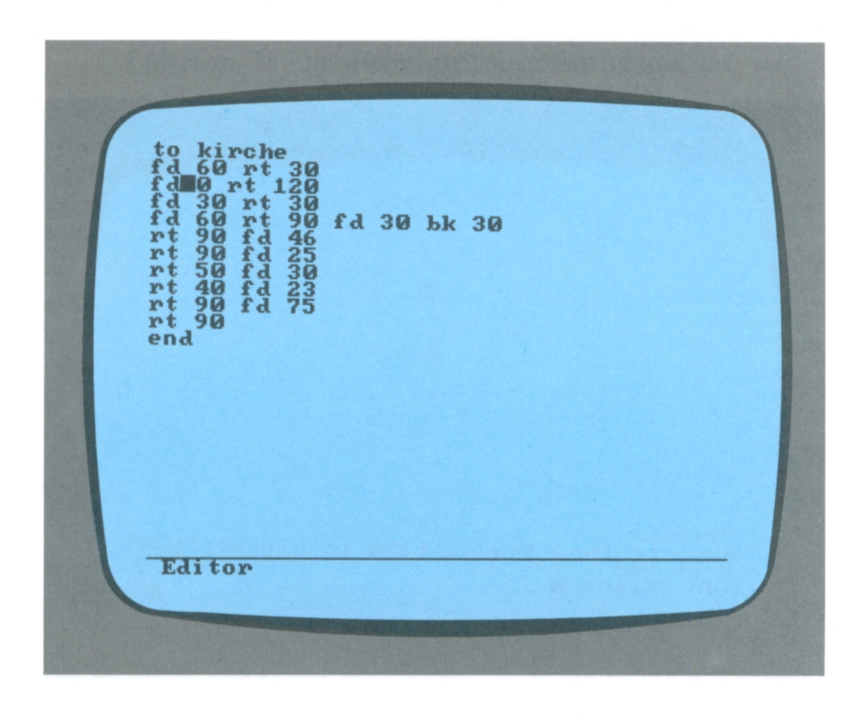
Lektion 7: Der Programmänderungsdienst



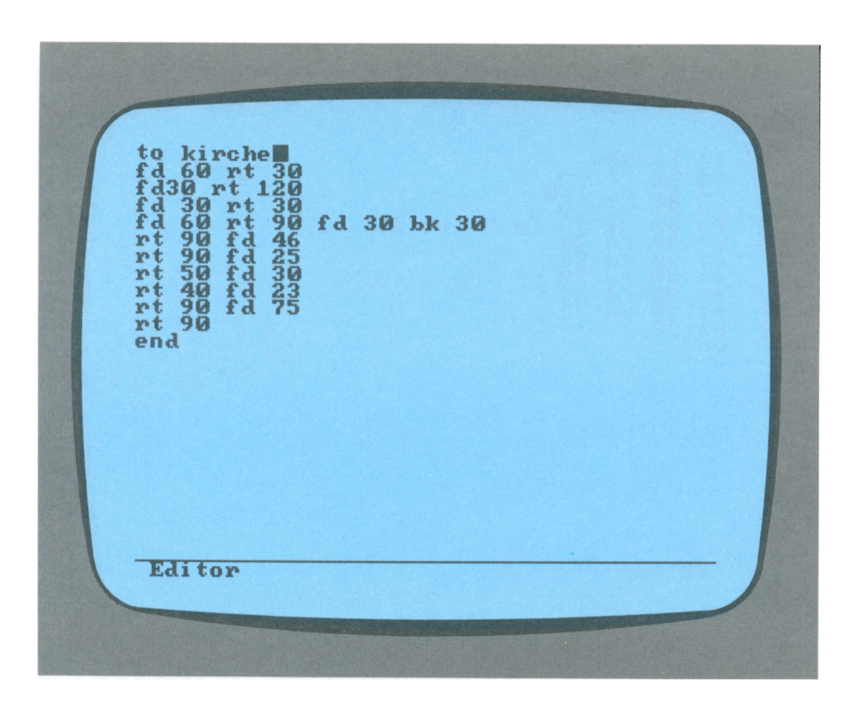
Wie oft werden wir noch erleben, daß ein erstelltes Programm nicht wie gewünscht abläuft, sondern mit einer Fehlermeldung endet. Im Beispiel haben wir unser Programm kirche eingetippt. Nach einem ersten Strich meckert der Computer. Sinngemäß übersetzt bedeutet der Fehlerkommentar: «Ich weiß nicht, wie fd30 in kirche gemacht wird.» Oder besser noch: «fd30 in kirche ist mir nicht erklärt worden!»



Tippen wir die Anweisung ed "kirche ein.



Nach dem Drücken der ENTER-Taste ändert sich der Bildschirm. In der Kopfzeile steht das schon bekannte to kirche. In der Fußzeile finden wir den Hinweis Editor. Der Bildschirm enthält kein Promptzeichen. Der Cursor steht rechts oben in der Schirmecke. Das Wort «Editor» könnte frei mit Redakteur gekennzeichnet werden, da ein Zeitungsredakteur Artikel (Programme) neu schreibt, verändert oder korrigiert. Mit ed wird also der Programmänderungsdienst (Editor) aufgerufen.



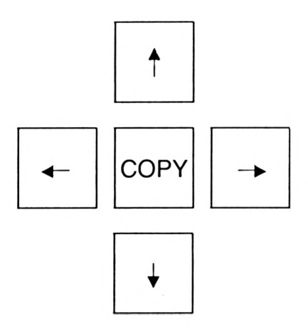
Unseren Tippfehler finden wir rasch. In der dritten Zeile haben wir zwischen fd und 30 ein Leerzeichen vergessen. Wir müssen jetzt den Cursor mit den Pfeiltasten genau über die Ziffer 3 stellen und die Leertaste drücken. Schon ist der Schaden behoben (vgl. nebenstehendes Bild).

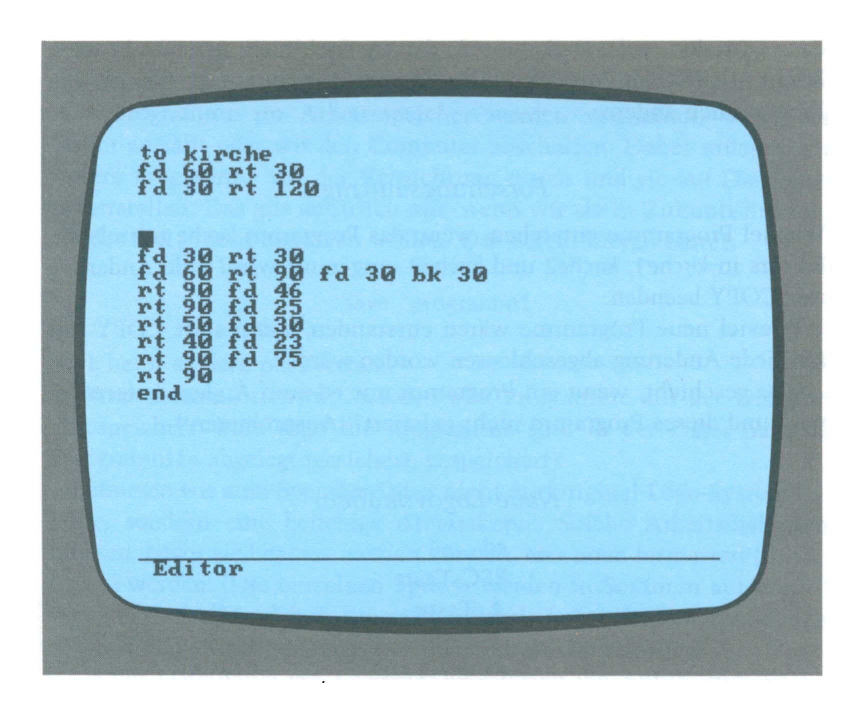
Tippfehler können wir vor dem Drücken der ENTER-Taste in gewohnter Weise korrigieren. Wer das noch einmal genau wissen will, sollte Lektion 4 erneut durchlesen. Bevor die Änderung beendet wird, testen wir alle vier Pfeiltasten und lassen den Cursor einmal rauf- und runtertanzen und von links nach rechts und zurückwandern. Nutzen wir doch die Tastenwiederholfunktion aus, indem wir sie länger gedrückt halten.

Mit CTRL-A und CTRL-E springt man an den Anfang und das Ende der jeweiligen Zeile. Löschen und Zeicheneinfügungen erfolgen wie gehabt.

```
to kirche
fd 60 rt 30
fd █0 rt 120
fd 30 rt 30
fd 60 rt 90 fd 30 bk 30
rt 90 fd 46
rt 90 fd 25
rt 50 fd 30
rt 40 fd 23
rt 90 fd 75
rt 90
end

 Editor
```





Doch wie fügt man neue Zeilen ein? Das zeigen wir an einem Beispiel. Gehen wir mit dem Cursor in die dritte Zeile und springen dann ans Zeilenende. Jetzt drücken wir einige Male die ENTER-Taste. Es entstehen entsprechend viele Leerzeilen, die mit Inhalten gefüllt werden können. Die ENTER-Taste erzeugt ein nicht sichtbares Zeichen, das trotzdem mit einer Stelle auf dem Schirm vorliegt. Dieses ENTER-Zeichen bewirkt eine Zeilenschaltung. Genauso können wir Zeilen zusammenschieben, indem wir jeweils mit CTRL-A an den Anfang einer Zeile gehen und dann die DEL-Taste drücken.

Solch eine Änderungssitzung beenden wir durch Drücken der COPY-Taste. Die COPY-Taste ist somit die Ende-Taste für den Editor. Nach dem Drücken der COPY-Taste ist der Bildschirm gelöscht, und in der ersten Zeile steht die Bestätigung.

Auch mit der ESC-Taste können wir den Editor verlassen. Doch das ist ein gewaltsamer Abbruch, und alle gemachten Änderungen sind vernichtet. Das Beenden mit COPY erzeugt ein neues Programm. Wer

das verhindern will, weil seine Änderung doch nicht gewünscht wird, bricht mit ESC ab. Mit dem Editor können wir ebenso die Namen von Programmen ändern.

Forschungsauftrag

Wieviel Programme entstehen, wenn das Programm kirche mittels des Editors in kirche1, kirche2 und kirche3 umgetauft wird? Jede Änderung mit COPY beenden.

Wieviel neue Programme wären entstanden, wenn statt COPY mit ESC jede Änderung abgeschlossen worden wäre?

Was geschieht, wenn ein Programm mit ed zum Ändern aufgerufen wird und dieses Programm nicht existiert? (Ausprobieren!)

Neue Logovokabeln

ed
ESC-Taste
↑ -Taste
↓ -Taste
COPY-Taste

Kurzform der Lektion

Das Ändern von Programmen geschieht mit dem Editor. Der Änderungsmodus oder Editierbetrieb wird durch den ed-Befehl erreicht. Mit den Pfeiltasten kann der Cursor an beliebige Stellen des abgebildeten Programms positioniert werden. Auch ans Zeilenende oder an den Zeilenanfang kommt man wie üblich. Mit COPY wird der Editor verlassen. ESC bricht eine Änderung ab, ohne daß sie ausgeführt wird. Das Ändern von Programmnamen und das Einfügen neuer Zeilen wird beschrieben. Auch die Neuerstellung von Programmen ist mit ed möglich.

Lektion 8: Programme sichern, laden, listen

Alle Programme im Arbeitsspeicher werden vernichtet, wenn der Strom ausfällt oder wir den Computer abschalten. Daher müssen wir unsere Programme vor der Vernichtung retten und sie auf Disketten sicherstellen. Das gilt natürlich nur, wenn wir sie in Zukunft hin und wieder mal erneut benutzen wollen. Der Befehl hierzu lautet:

```
save "programm1
```

Save heißt sichern oder retten.

Unter diesem Stichwort werden alle Programme auf der Diskette abgespeichert. Man sagt, die Programme sind in der Datei namens «programm1» abgelegt (gesichert, gespeichert).

Benutzen wir zum Speichern aber nicht die Original-Logo-Systemdiskette, sondern eine beliebige Arbeitskopie. Solche Arbeitsdisketten müssen, bevor sie benutzt werden können, erst noch formatiert (initialisiert) werden. (Die einzelnen Spuren werden in Sektoren aufgeteilt.) Wer keine zur Hand hat, möge im mitgelieferten Bedienerhandbuch von Schneider nachschlagen und sich eine solche erstellen.

Da der Dateiname ein Logowort ist, dürfen die Anführungsstriche nicht vergessen werden.

Im Inhaltsverzeichnis der Diskette finden wir den Eintrag «programm1» wieder. Das Inhaltsverzeichnis einer Diskette erhalten wir auf dem Schirm durch folgenden Befehl aufgelistet:

```
dir
```

dir ist die Abkürzung von directory und bedeutet Inhaltsverzeichnis.

Wer seine Programme auf Papier ausgedruckt haben will, muß enttäuscht werden. Es gibt keine Logovokabel, die einen Drucker anspricht. Wir können nichts auf Papier drucken. Das geht nur von CP/M aus (vgl. Lektionsende).

Haben wir Programme auf Disketten gespeichert, so können wir sie jederzeit in den Arbeitsspeicher laden (einlesen) und wie gewohnt benutzen. Das Einlesen geschieht mit dem Befehl:

```
load "programm1
```

Mit dem save-Befehl wird immer der gesamte Inhalt des Arbeitsspeichers gesichert, also auch jede Programmleiche und sonstiger Schrott. Vor dem Sichern von Programmen müssen wir uns also den Inhalt des Arbeitsspeichers mit pots ansehen. Unnütze Programme sollten dann gelöscht werden. Gelöscht wird mit dem Befehl er. Erase heißt ausradieren oder löschen. Ein Einzelprogramm löschen wir mit

er "sonne

Wollen wir mehrere Programme löschen, zählt man alle Programme auf und packt die Namen in eckige Klammern.

er [sonne probe kirche]

Soll der gesamte Arbeitsspeicher leergefegt werden, erledigt das der Befehl

bye

Doch mit bye haben wir jetzt die Betriebssystemebene erreicht. Wir müssen Logo neu laden; einfach logo ⟨ENTER⟩ eintippen. Eine weitere Möglichkeit bietet die Anweisung

er glist ".DEF

Soll eine Programmdatei auf dem Datenträger gelöscht werden, so werden wir erneut von Schneider-Logo im Stich gelassen.

Um Logo-Programme aufzulisten, müssen wir das Betriebssystem CP/M aktivieren.

bye ⟨ENTER⟩

Jetzt schalten wir mit CTRL-P den Drucker ein. Um die Datei «programm1» aufzulisten, tippen wir ein

TYPE PROGRAMM1.LOG ⟨ENTER⟩

Mit erneutem CTRL-P schalten wir den Drucker ab.
Eine Datei wird gelöscht durch den CP/M-Befehl ERA.

ERA PROGRAMM1.LOG ⟨ENTER⟩

Logo wird erneut von der Systemdiskette geladen mit

logo ⟨ENTER⟩

Hinweis: Beim Editieren von eingelesenen Programmdateien diese unter einem *neuen* Namen auf der Diskette speichern, dann die alte Version auf der Diskette löschen.

Forschungsaufträge

Was passiert, wenn wir auf der Diskette Programme sichern wollen und einen Dateinamen wählen, der schon existiert?
Bei den Schreib- und Lesebefehlen save und load mußten wir den Namen Anführungsstriche voranstellen. Was passiert, wenn wir sie vergessen?

Neue Logovokabeln

save
load
dir
er
CP/M-Befehle:
ERA
CTRL-P
TYPE

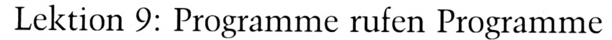
Kurzform der Lektion

Lektion 8 beschreibt das Sichern von Arbeitsspeicherinhalten (Programmen) auf Disketten und das Einlesen (Laden) von Programmen von der Arbeitsdiskette. Bei den Befehlen save und load müssen den Namen Anführungsstriche vorangestellt werden. Kommt der gewählte Datei-

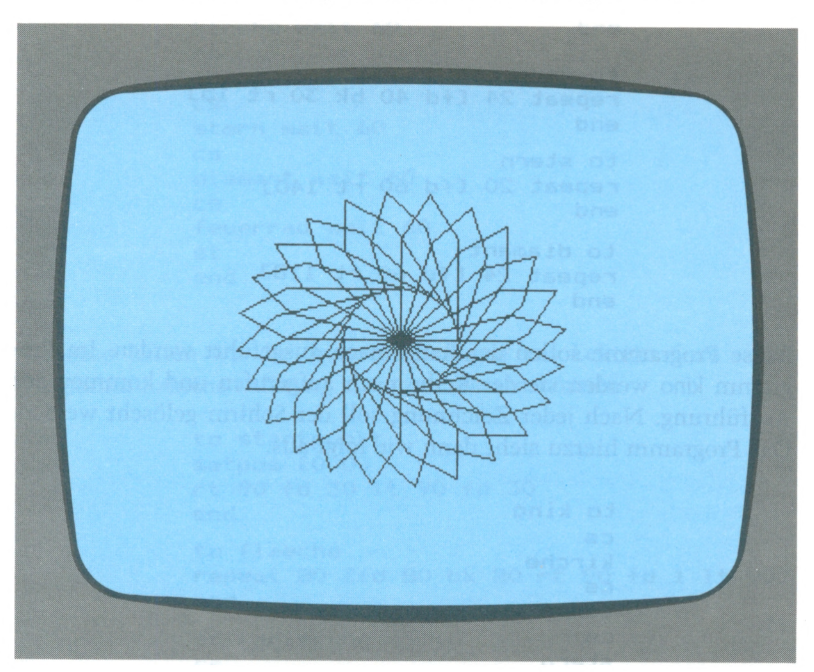
name bereits auf der Diskette vor, läßt sich diese Datei nicht einfach mit save überschreiben (... already exists wird als Fehlermeldung ausgegeben). Will man eine bestehende Datei überschreiben, muß diese unter CP/M auf der Diskette gelöscht werden.

Mit dir sieht man sich das Inhaltsverzeichnis einer Diskette an.

Programme lassen sich nur auf dem Umweg über CP/M auf einem Drucker listen. Befehle zum Ausmisten des Arbeitsspeichers (Löschbefehle) werden vorgestellt.



Lektion 9: Programme rufen Programme



In den bisherigen Programmen haben wir unter einem frei gewählten Namen beliebige Logovokabeln zusammengefaßt. Wir können genausogut unsere eigenen Sprachschöpfungen in den Programmen benutzen. Betrachten wir die folgenden bereits bekannten Programme:

```
to kirche
fd 60 rt 30
fd 30 rt 120
fd 30 rt 30
fd 60 rt 90 fd 30 bk 30
rt 90 fd 46
rt 90 fd 25
rt 50 fd 30
rt 40 fd 23
rt 90 fd 74
rt 90
end
```

```
to sonne
repeat 36 [fd 80 bk 80 rt 10]
end

to feuerrad
repeat 24 [fd 40 bk 30 rt 15]
end

to stern
repeat 20 [fd 60 rt 140]
end

to diamant
repeat 24 [fd 60 rt 118]
end
```

Diese Programme sollen der Reihe nach ausgeführt werden. Im Programm kino werden sie der Reihe nach aufgerufen und kommen zur Ausführung. Nach jeder Zeichnung soll der Schirm gelöscht werden. Das Programm hierzu sieht dann wie folgt aus.

```
to kino
cs
kirche
cs
sonne
cs
stern
cs
diamant
cs
feuerrad
end
```

Das einzelne Bild möchten wir vielleicht einen Augenblick länger betrachten. Dazu müßte der Computer vor Ausführung des nächsten Programms in kino eine Pause machen. Lassen wir ihn jeweils eine Sekunde warten. Der Befehl wait 60 läßt den Computer einige Sekunden lang warten. wait heißt warten. Die Zahlenangabe für wait kann eine beliebige ganze Zahl sein. Die Wartezeit beträgt das 0,22fache der Ganzzahl. Verstecken wir auch die Schildkröte, so daß wir nur die entstehenden Striche sehen. Der Befehl ht (Abkürzung von hide turtle, was «verstecke Schildkröte» heißt) bewirkt es. Das Gegenteil hierzu wäre st (Abkürzung von show turtle, das bedeutet «zeige Schildkröte»).

```
to kino
cs ht
kirche wait 60
cs
sonne wait 60
cs
stern wait 60
cs
diamant wait 60
cs
feuerrad wait 60
st
end
```

Genauso hätten wir die drei Anweisungen aus Lektion 6 zu einem Programm namens radierung zusammenfassen können.

```
to startpunkt
setpos [0 0]
rt 90 fd 30 lt 90 fd 30
end

to flaeche
repeat 80 [fd 80 bk 80 rt 90 fd 1 lt 90]
end

to radierung
cs
flaeche
startpunkt
pe stern pd
end
```

Der Befehl setpos [0 0] positioniert die Schildkröte in der Schirmmitte (vgl. auch Lektion 12).

Nachfolgendes Programm ist verblüffend. Es ruft nicht nur unser bekanntes kino auf, sondern sich selbst.

```
to dauerkino
kino
dauerkino
end
```

Das Programm wird immer wieder ausgeführt. Um das Programm dauerkino zu beenden, müssen wir gewaltsam mit CTRL-G oder beque-

mer mit ESC den Abbruch erzwingen. Wir hätten den Dauereffekt auch erreicht, wenn wir im Programm kino die neue Schlußzeile kino eingefügt hätten.

```
to kino
cs ht
kirche wait 60
cs
sonne wait 60
cs
stern wait 60
cs
diamant wait 60
cs
feuerrad wait 60
st
kino
end
```

Dieser Dauereffekt hätte auch mit einem Wiederholungsbefehl erreicht werden können:

```
repeat 100 [kino]
```

Auf gleiche Weise lassen sich Drehfiguren erzeugen. Sehen wir uns das Programm turm an, das den Turm unserer altehrwürdigen Kirche zeichnet.

```
to turm
fd 60 rt 30
fd 30 rt 120
fd 30 rt 30
fd 60 rt 90
fd 30 rt 90
end
```

Durch eine Drehung und anschließendes Aufrufen des eigenen Programmnamens entsteht das schöne Ornament zu Anfang dieses Kapitels.

```
to drehturm
fd 60 rt 30
fd 30 rt 120
fd 30 rt 30
```

```
fd 60 rt 90
fd 30 rt 90
rt 15
drehturm
end
```

Forschungsauftrag

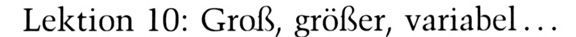
Ein Zwölfeck mit der Seitenlänge 30 (vgl. Lektion 5) soll durch Wiederholung wie beim Beispiel drehturm ein Ornament bilden.

Neue Logovokabeln

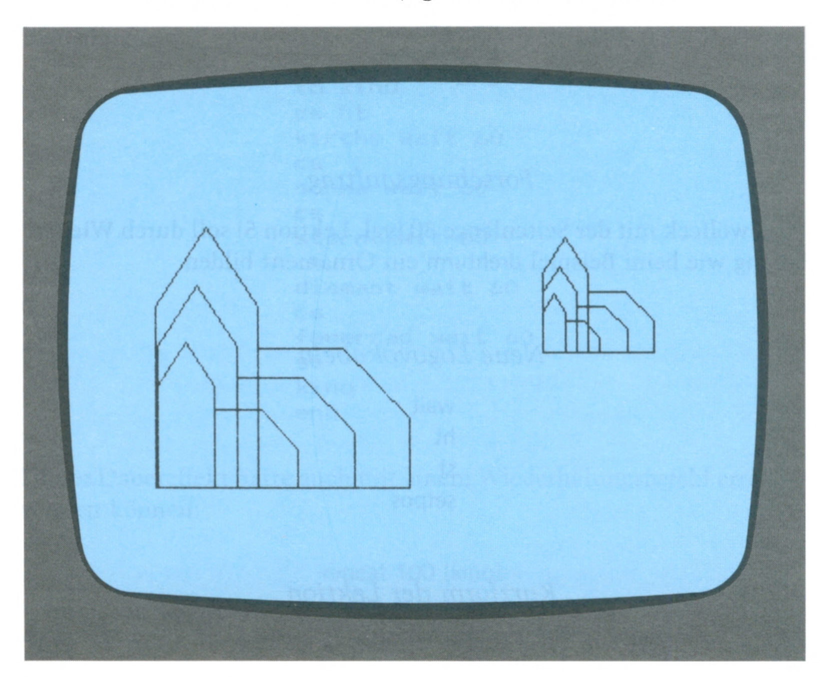
wait
ht
st
setpos

Kurzform der Lektion

Lektion 9 zeigt, wie Programme beliebig andere vom Benutzer definierte Programme aufrufen können. Selbst das Aufrufen des eigenen Programmnamens ist möglich. Solche sich selbst aufrufenden Programme sind endlos. Sie können nur mit der ESC-Taste unterbrochen werden. Bequem können auf diese Weise Drehfiguren oder andere «Dauerprogramme» realisiert werden. Solche Programme ließen sich ebenso durch eine Wiederholungsanweisung mit 100 Wiederholungen darstellen.



Lektion 10: Groß, größer, variabel…



In den vorangegangenen Abschnitten haben wir eine Fülle von Figuren durch Ändern von Zahlenwerten entstehen lassen. Wenn wir bei einem Vorwärtsbefehl oder einer Drehung einen anderen Zahlenwert als Eingabe wünschten, mußten wir die jeweilige Anweisungszeile mit dem Logo-Editor ändern. Auch diesen Aufwand können wir durch einen kleinen Trick vermindern.

Statt einer genau vorgegebenen Zahl benutzen wir jetzt Platzhalter. Man nennt solche Platzhalter auch Variable oder Parameter. Sehen wir uns ohne viel Gerede ein solches Programm an.

```
to rahmen :schritte
repeat 4 [fd :schritte rt 90]
end
```

Das Programm rahmen ist genauso aufgebaut wie die Programme aus Lektion 6. In der Kopfzeile finden wir neben dem Namen des Programms den Platzhalter :schritte. Solche Programmvariable sind immer am Doppelpunkt erkennbar. In der Wiederholungsanweisung finden wir den Befehl fd :schritte. Das sagt ganz einfach, daß die Schildkröte soundso viele :schritte vorwärts marschieren soll. Die Variable :schritte ist zahlenmäßig noch unbekannt. Lassen wir jetzt solch ein Programm ablaufen. Wir müssen den Programmnamen eintippen, dem eine aktuelle Zahl für die Variable folgt.

```
rahmen 50
```

Dieses Programm rahmen können wir für beliebig große Quadrate benutzen. Probieren wir's ein paarmal aus. Geben wir unter Zuhilfenahme von CTRL-Y ein:

```
rahmen 90
rahmen 70
rahmen 30
```

Vergessen wir einmal, beim Eintippen eine Zahl für eine Programmvariable anzugeben, so führt das zu einer Fehlermeldung.

```
rahmen 〈ENTER〉
not enough inputs to rahmen
```

Wir können ebenfalls den Zufallszahlengenerator benutzen, der uns Werte für die Programmvariable erzeugt.

```
rahmen random 80
```

Wenn wir diese Anweisung noch in eine repeat-Anweisung einbinden, können wir schnell eine Fülle kleiner und großer Quadrate erzeugen, die alle vom selben Programm ausgeführt werden.

```
repeat 10 [rahmen random 80]
```

Auf gleiche Weise lassen sich verschieden große Dreiecke, Sechsecke und Zwölfecke erzeugen.

```
to dreieck :schritte
repeat 3 [fd :schritte rt 120]
end

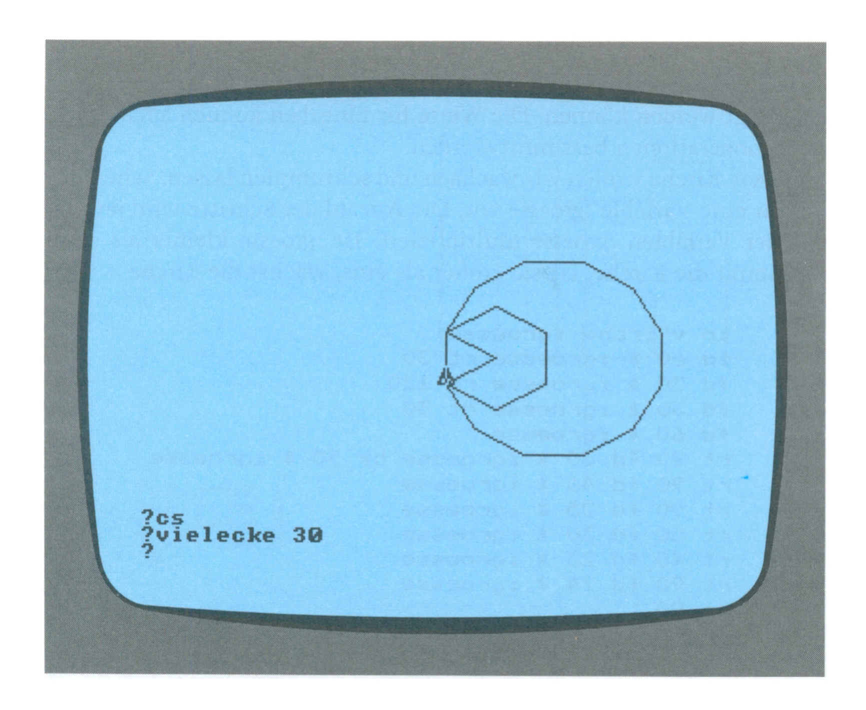
to sechseck :schritte
repeat 6 [fd :schritte rt 60]
end

to zwoelfeck :schritte
repeat 12 [fd :schritte rt 30]
end
```

Natürlich lassen sich auch Programme aus Programmen aufrufen, wobei diese Leitprogramme ebenfalls Programmvariable aufweisen können. Erstellen wir ein Programm namens vielecke mit der Variablen :groesse, die die Schritte der aufzurufenden Programme steuert.

```
to vielecke :groesse
dreieck :groesse
sechseck :groesse
zwoelfeck :groesse
end
```

Achten wir genau darauf, was hier passiert. Die Variable des Programms dreieck heißt :schritte. Beim Benutzen unseres Programms dreieck wird als aktueller Wert die Variable :groesse eingesetzt!
Lassen wir das Programm vielecke 30 ablaufen.



Als nächstes wollen wir unsere Sternprogramme variabel gestalten und die Drehung über einen Platzhalter steuern.

```
to varistern1 :winkel
repeat 20 [fd 60 rt :winkel]
end
```

Natürlich kann ein Programm mehr als eine Variable enthalten. Das Folgeprogramm enthält eine weitere Variable für die Länge des Strichs.

```
to varistern2 :schritte :winkel
repeat 20 [fd :schritte rt :winkel]
end
```

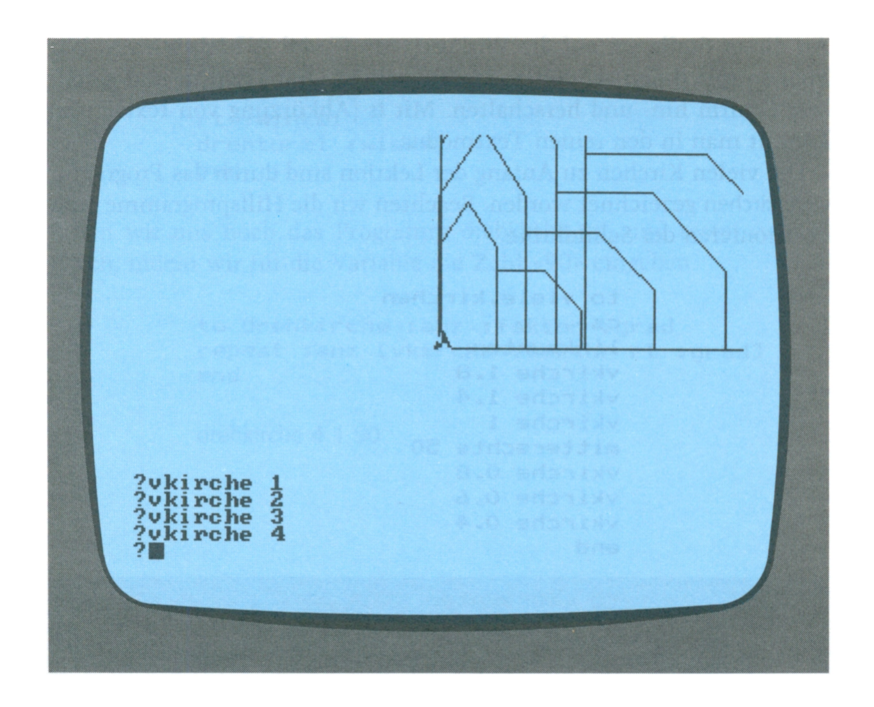
Das Beispiel varistern3 enthält eine weitere Variable zur Steuerung der Anzahl der Wiederholungen für den repeat-Befehl.

```
to varistern3 :anz :schritte :winkel
repeat :anz [fd :schritte rt :winkel]
end
```

In den folgenden Beispielen wollen wir zeigen, daß die Werte für die Variablen nicht nur durch eine Zahlenvorgabe oder random beim Ablauf festgelegt werden können. Die Werte für Eingaben können auch durch Rechenoperationen bestimmt werden.

Unsere Kirche wollen wir wachsen und schrumpfen lassen. Sehen wir hierfür eine Variable :groesse vor. Die Anzahl der Schritte wird jeweils mit der Variablen :groesse multipliziert. Ist :groesse kleiner als eins, schrumpft die Kirche, ist sie größer als eins, wächst die Kirche.

```
to vkirche :groesse
fd 60 * :groesse rt 30
fd 30 * :groesse rt 120
fd 30 * :groesse rt 30
fd 60 * :groesse
rt 90 fd 30 * :groesse bk 30 * :groesse
rt 90 fd 46 * :groesse
rt 90 fd 25 * :groesse
rt 50 fd 30 * :groesse
rt 40 fd 23 * :groesse
rt 90 fd 74 * :groesse
rt 90
end
```



Testen wir einige Aufrufe von vkirche mit den Werten 1, 2, 3 und 4. Der Schirm ist zu klein. Der Computer produziert dennoch das, was auf den Schirm paßt, obwohl manches über die Schirmgrenzen hinausgeht.

Wiederholen wir das noch einmal, geben aber vorher den Befehl wrap ein. Die Schildkröte zeichnet praktisch ihre Spur «um den Schirm hinten herum» weiter und kommt dann unten auf dem Schirm an. Wollen wir diesen Effekt unterdrücken, tippen wir den Befehl window ein. Window heißt «Fenster» oder «Ausschnitt». Mit wrap kommen wir wieder in den alten Zustand zurück. Wrap heißt umwickeln, womit das rückwärtige Um-den-Schirm-herum-Zeichnen gemeint ist. wrap bewirkt oft interessante Effekte, wenn Striche über die Schirmgrenzen hinausgehen.

An dieser Stelle sei auf die Wirkung von fs und ss hingewiesen. Wir können mit ihnen schnell zwischen vollem Grafikschirm und geteiltem Schirm hin- und herschalten. Mit ts (Abkürzung von textscreen) kommt man in den reinen Textmodus.

Die vielen Kirchen zu Anfang der Lektion sind durch das Programm viele.kirchen gezeichnet worden. Beachten wir die Hilfsprogramme zum Positionieren der Schildkröte.

```
to viele.kirchen
cs
linksunten
vkirche 1.8
vkirche 1.4
vkirche 1
mitterechts 50
vkirche 0.8
vkirche 0.6
vkirche 0.4
end

to linksunten
pu
setpos [0 0]
bk 80 lt 90 fd 155 rt 90
pd
end

to mitterechts :schritte
pu setpos [0 0]
rt 90
fd :schritte
lt 90
pd
end
```

Natürlich klappt diese Variablengeschichte auch bei Programmen, die sich selbst aufrufen. Testen wir das einmal an schon bekannten Drehprogrammen. Sehen wir dabei für den Drehwinkel eine Variable vor. Nennen wir das Programm drehturm1.

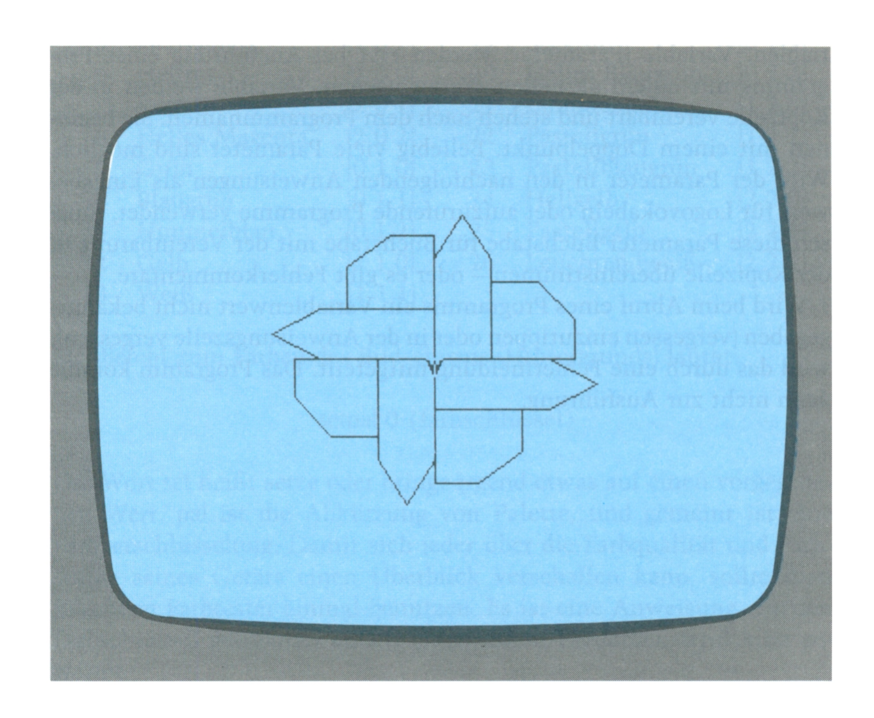
```
to drehturm1 :winkel
fd 60 rt 30
fd 30 rt 120
```

```
fd 30 rt 30
fd 60 rt 90
fd 30 rt 90
rt :winkel
drehturm1 :winkel
end
```

Sehen wir uns noch das Programm drehkirche an, und testen wir es gleich, indem wir für die Variable die Zahl «90» eingeben.

```
to drehkirche :anz :faktor :grad
repeat :anz [vkirche :faktor rt :grad]
end
```

drehkirche 4 1 90



Forschungsaufträge

Was passiert, wenn wir in den Programmen die Ziffer «0» für die verschiedenen Variablen eingeben? (Nicht gleichzeitig mehrere Nullen eingeben, damit die Wirkung beobachtet werden kann.)

Welche Fehlermeldung entsteht, wenn im Programm rahmen in der Wiederholungsanweisung bei fd :schritte der Buchstabe «e» versehentlich nicht eingetippt worden wäre (ändern und testen)?

Neue Logovokabeln

*
window
wrap

Kurzform der Lektion

Lektion 10 erklärt den Nutzen und die Verwendung von Programmvariablen. Variable (Parameter) werden erst bei Ausführung eines Programms mit einem aktuellen Wert versehen. Variable werden in der Kopfzeile vereinbart und stehen nach dem Programmnamen. Sie beginnen mit einem Doppelpunkt. Beliebig viele Parameter sind möglich. Wird der Parameter in den nachfolgenden Anweisungen als Eingabewert für Logovokabeln oder aufzurufende Programme verwendet, müssen diese Parameter Buchstabe für Buchstabe mit der Vereinbarung in der Kopfzeile übereinstimmen – oder es gibt Fehlerkommentare.

Wird beim Abruf eines Programms ein Variablenwert nicht bekanntgegeben (vergessen einzutippen oder in der Anweisungszeile vergessen), wird das durch eine Fehlermeldung mitgeteilt. Das Programm kommt dann nicht zur Ausführung.

Lektion 11: Farbe ist Trumpf

Schneider-Logo bietet eine ganze Farbpalette an. Wir können den Bildschirm färben. Ebenso lassen sich Striche in verschiedenen Farben ziehen. Vier verschiedene Farben können auf dem Schirm gleichzeitig erscheinen. Neben der Hintergrundfarbe kann mit maximal drei Farben gezeichnet werden. Doch die Kennzahlen aus dem BASIC-Handbuch müssen in Logo als Liste verschlüsselt werden. (Genaugenommen wird die Kennzahl in ein Stellenwertsystem mit der Basis «3» umgerechnet.)

Die 27 Farben sind:

Nr.	Farbe	Schlüssel	Nr.	Farbe	Schlüssel
0	Schwarz	[0 0 0]	14	Pastellblau	[1 1 2]
1	Blau	[0 0 1]	15	Orange	[2 1 0]
2	Hellblau	[0 0 2]	16	Rosa	[2 1 1]
3	Rot	[1 0 0]	17	Pastellmagenta	[2 1 2]
4	Magenta	[1 0 1]	18	Hellgrün	[0 2 0]
5	Hellviolett	[1 0 2]	19	Seegrün	[0 2 1]
6	Hellrot	[2 0 0]	20	Helles Blaugrün	[0 2 2]
7	Purpur	[2 0 1]	21	Limonengrün	[1 2 0]
8	Helles Magenta	[2 0 2]	22	Pastellgrün	[1 2 1]
9	Grün	[0 1 0]	23	Pastellblaugrün	[1 2 2]
10	Blaugrün	[0 1 1]	24	Hellgelb	[2 2 0]
11	Himmelblau	[0 1 2]	25	Pastellgelb	[2 2 1]
12	Gelb	[1 1 0]	26	Leuchtendweiß	[2 2 2]
13	Weiß	[1 1 1]			

Der Befehl zum Färben des Bildschirms (Hintergrunds) lautet:

setpal 0 ⟨farbschlüssel⟩

Das Wort set heißt setze oder bringe irgend etwas auf einen vorgegebenen Wert. pal ist die Abkürzung von Palette, und gemeint ist eine Farbverschlüsselung. Damit sich jeder über die Farbqualität und Farbstufen seines Geräts einen Überblick verschaffen kann, sollte man folgenden Farbtester einmal benutzen. Es ist eine Anweisung, die den Farbschlüssel per Zufall bildet. Nach jedem Umfärben wird kurz gewartet.

Im Anhang ist eine nützliche Routine color, die eine der obigen Nummern in einen Farbschlüssel umrechnet. Nützlich sind auch die dortigen Routinen farbtester und farbauswahl.

repeat 50 [setpal 0 farbe wait 20]

```
to farbe
op ( se random 3 random 3 random 3 )
end
```

Die Operation farbe ist recht nützlich zum Erzeugen eines Farbschlüssels und wird in Lektion 12 erneut eingesetzt. Wer systematisch alle Farben von 0 bis 26 sehen will, sollte das Programm farbtester aus dem Anhang entnehmen.
Wählen wir uns aus den soeben gesehenen Farben Rot und Gelb heraus, um schnell ein Wetterleuchten zu simulieren.

```
to wetterleuchten
setpal 0 [0 0 0] wait 20
setpal 0 [1 0 0] wait 10
setpal 0 [0 0 0] wait 20
setpal 0 [1 1 0] wait 2
setpal 0 [0 0 0] wait 1
setpal 0 [1 1 0] wait 2
end
```

Binden wir dieses Programm in eine repeat-Anweisung ein, läßt sich der Effekt beliebig oft wiederholen.

Als nächstes zeichnen wir mit drei Farben gleichzeitig. Die drei Stifte werden mit den Ziffern 1, 2 und 3 angesprochen. Um mit einem dieser Stifte zu zeichnen, müssen wir jeweils vor einem Grafikbefehl bestimmen, welcher Stift jetzt dran ist. Das geschieht mit dem Befehl

setpc ⟨stiftnummer⟩

setpc ist die Abkürzung von setpencolor und heißt «setze Stiftfarbe fest» für den angegebenen Stift.

Benutzen wir zum Zeichnen als nächstes die Stiftfarben Rot, Gelb und Blau. Bevor wir überhaupt zeichnen können, müssen wir noch die Farben für jeden Stift festlegen.

```
cs
setpal 0 [0 0 0]
setpal 1 [2 0 0]
setpal 2 [1 1 0]
setpal 3 [0 0 2]
```

Zeichnen wir drei farbige Sonnen auf schwarzem Hintergrund. Wandern wir mit der Schildkröte an den linken Schirmrand und tippen ein:

```
lt 90 fd 100 rt 180
septc 1 sonne 20
fd 50
septc 2 sonne 20
fd 50
septc 3 sonne 20
```

Die drei Sonnen können wir umfärben, indem wir jeweils andere Farbpaletten wählen:

```
setpal 1 [2 1 1]
setpal 2 [2 2 0]
setpal 3 [0 1 2]
```

Als nächstes zeichnen wir unseren altbekannten drehturm in den Farben Pastellblau, Orange und Rosa. Er soll auf schwarzem Hintergrund erscheinen. Also müssen wir die Hintergrundfarbe 0 vorsehen.

```
to drehturm.3f
cs
setpal 0 [0 0 0]
repeat 8 [setpc 1 turm rt 15 setpc 2 turm rt 15
          setpc 3 turm rt 15]
end
```

Jetzt sollten wir die bisherigen Farbbeispiele auf einem farbigen Untergrund wiederholen.

Forschungsauftrag

Testen wir einmal die Logovokabel pal mit den Eingaben 0 bis 3. Testen wir mittels des repeat-Befehls zigmal die Anweisung:

```
fd 10 setpal random 4 farbe setpc (1 + random 3)
```

Neue Logovokabeln

pal
setpal
setpc

Kurzform der Lektion

Lektion 11 beschreibt die Farbmöglichkeiten von Schneider-Logo. Der Schirm kann eingefärbt werden, zusätzlich kann mit drei Farbstiften auf dem Farbhintergrund gezeichnet werden. Es gibt 27 verschiedene Farben, die untersucht worden sind. Die Stiftfarben lassen sich beliebig umfärben. Beim Ändern einer Stiftfarbe ändern sich auch die vorher gezeichneten Striche dieses Stifts.

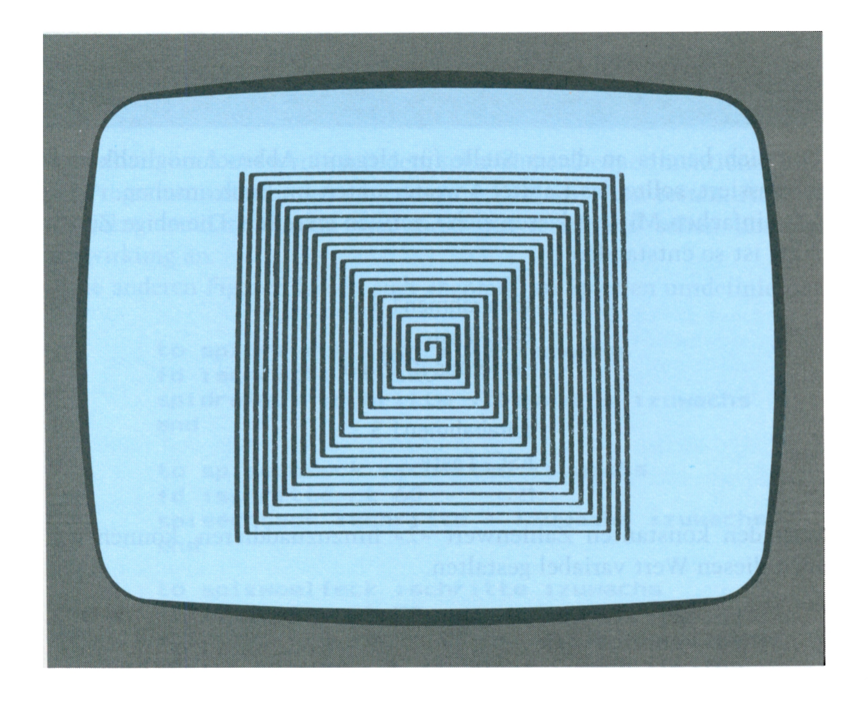
Lektion 12: Ideen und Experimente

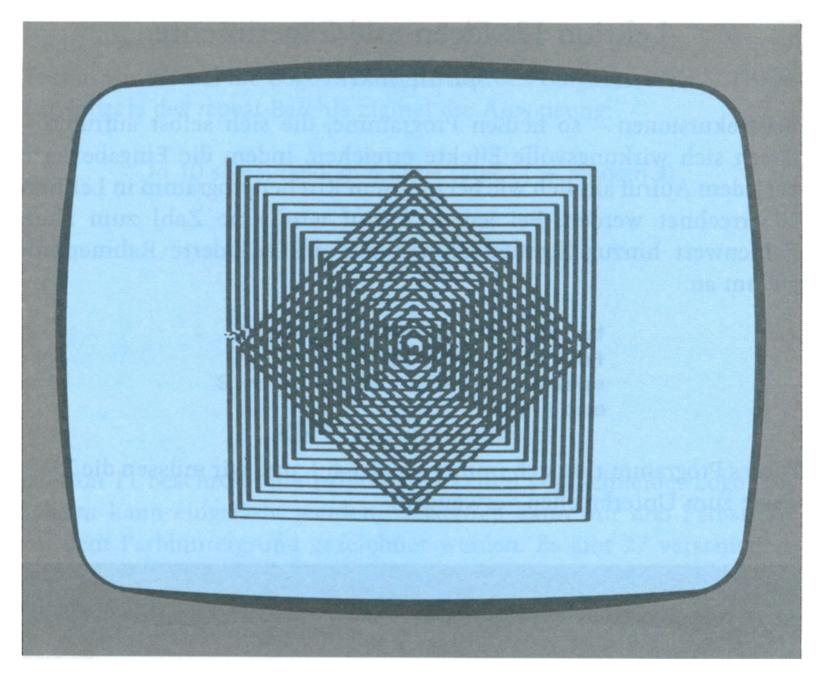
Spiraleffekte

Bei Rekursionen – so heißen Programme, die sich selbst aufrufen – lassen sich wirkungsvolle Effekte erreichen, indem die Eingabewerte bei jedem Aufruf ähnlich wie bei unserem Kirchenprogramm in Lektion 10 errechnet werden. Bei jedem Aufruf wird eine Zahl zum alten Zahlenwert hinzugefügt. Sehen wir uns das geänderte Rahmenprogramm an.

```
to spiralrahmen1 :schritt
fd :schritt rt 90
spiralrahmen1 :schritt + 2
end
```

Dieses Programm ruft sich immer wieder auf, und wir müssen die ESC-Taste zum Unterbrechen verwenden.



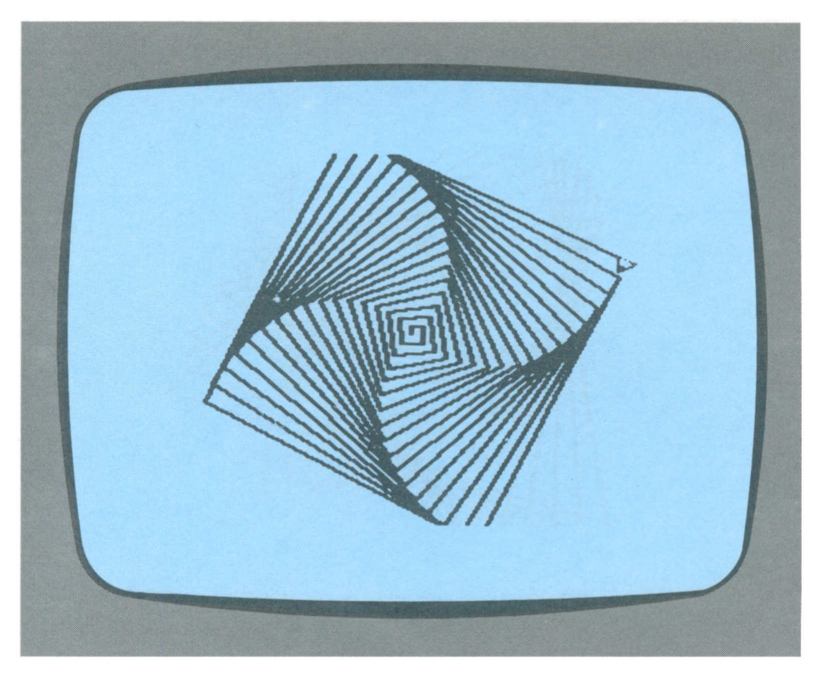


Wer sich bereits an dieser Stelle für elegante Abbruchmöglichkeiten interessiert, sollte das Kapitel 4 weiter unten im Buch ansehen.
Mit einfachen Mitteln lassen sich Grafiken erstellen. Die obige Zeichnung ist so entstanden:

```
cs
spiralrahmen1 3
ESC-Taste
pu setpos [0 0] pd
rt 45
spiralrahmen1 3
ESC-Taste
fs
```

Statt den konstanten Zahlenwert «2» hinzuzuaddieren, können wir auch diesen Wert variabel gestalten.

```
to spiralrahmen2 :schritt :zuwachs
fd :schritt rt 90
spiralrahmen2 :schritt + :zuwachs :zuwachs
end
```



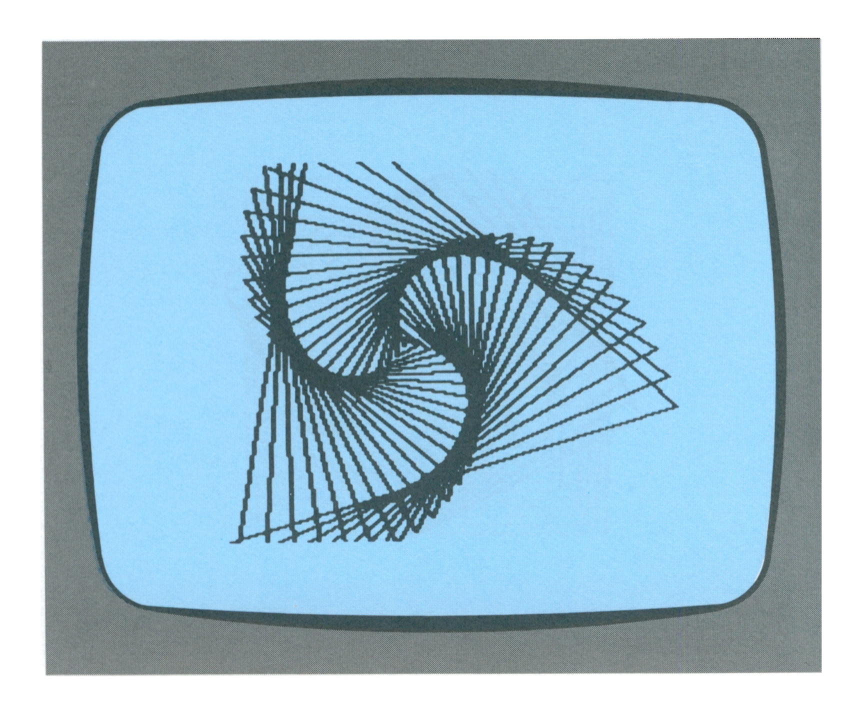
Wir sollten noch ein wenig mit dem spiralrahmen2 experimentieren und das Programm mit den verschiedensten Zahlenwerten testen. Ändern wir jetzt den Drehwinkel von rt 90 auf rt 89 und sehen uns die Auswirkung an.

Die anderen Figuren lassen sich ebenfalls zu Spiralen umdefinieren:

```
to spidreieck :schritte :zuwachs
fd :schritte rt 120
spidreieck :schritte + :zuwachs :zuwachs
end

to spisechseck :schritte :zuwachs
fd :schritte rt 60
spisechseck :schritte + :zuwachs :zuwachs
end

to spizwoelfeck :schritte :zuwachs
fd :schritte rt 30
spizwoelfeck :schritte + :zuwachs :zuwachs
end
```



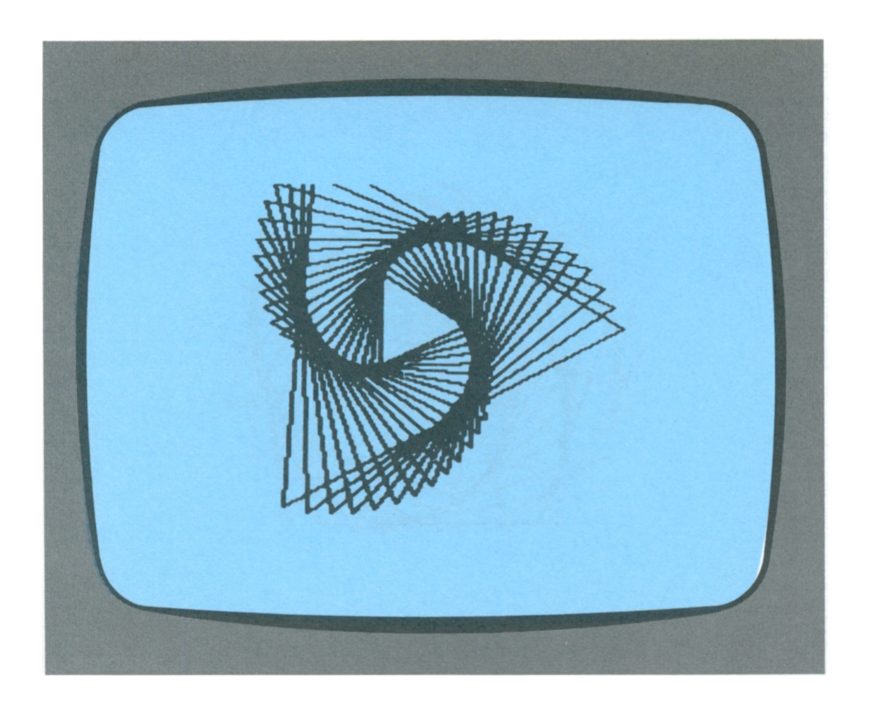
Ein hübscher Grafikeffekt entsteht, wenn wir in unserem spidreieck auch noch die Drehung veränderlich gestalten, indem wir statt 120 Grad Winkel vorsehen, die Werte mal über oder unter 120 haben.

```
to spisuper :schritte :zuwachs :grad
fd :schritte rt :grad
spisuper :schritte + :zuwachs :zuwachs :grad
end
```

Die obige Grafik ist mit dem Programm spisuper und den folgenden Variablenwerten entstanden.

```
cs
window
spisuper 3 3 122
```

Denken wir auch an die Möglichkeit, bei allen Experimenten die Auswirkung von wrap und window auszutesten.



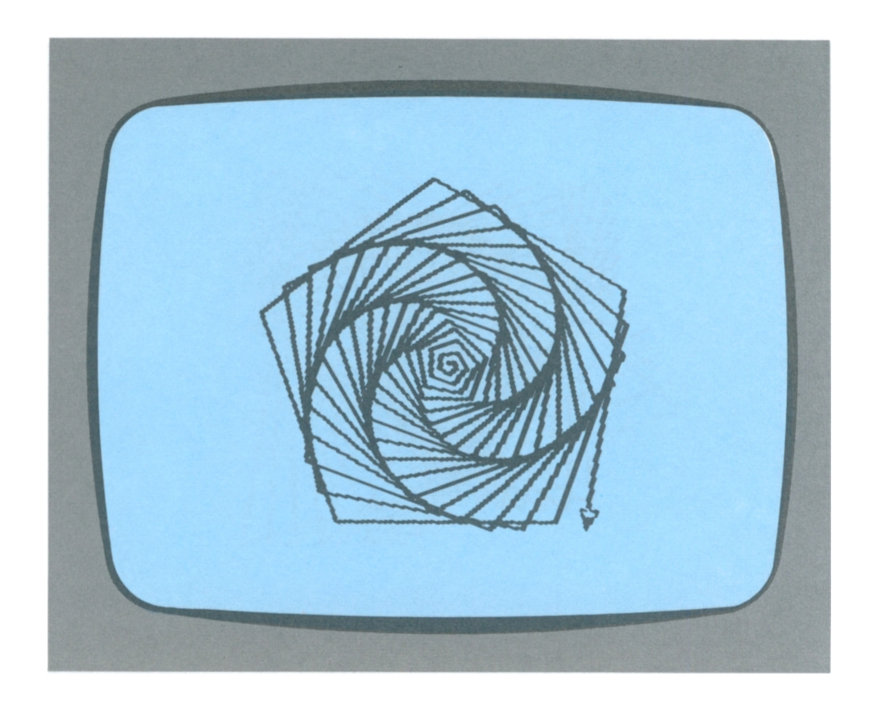
Experimentieren wir noch ein wenig und testen unser spisuper mit den Aufrufen:

cs
spisuper 40 2 122 (siehe obige Grafik)

cs
spisuper 1 1 70 (siehe S. 104)

cs
spisuper 2 2 89 (siehe S. 101)

Welche unterschiedlichen Figuren entstehen, wenn das Programm spisuper mit den unterschiedlichsten Drehwinkeln (Bereich von 0 bis 150 Grad) ausgeführt wird und für die Variable :zuwachs auch mal die Ziffer 0 eingegeben wird?

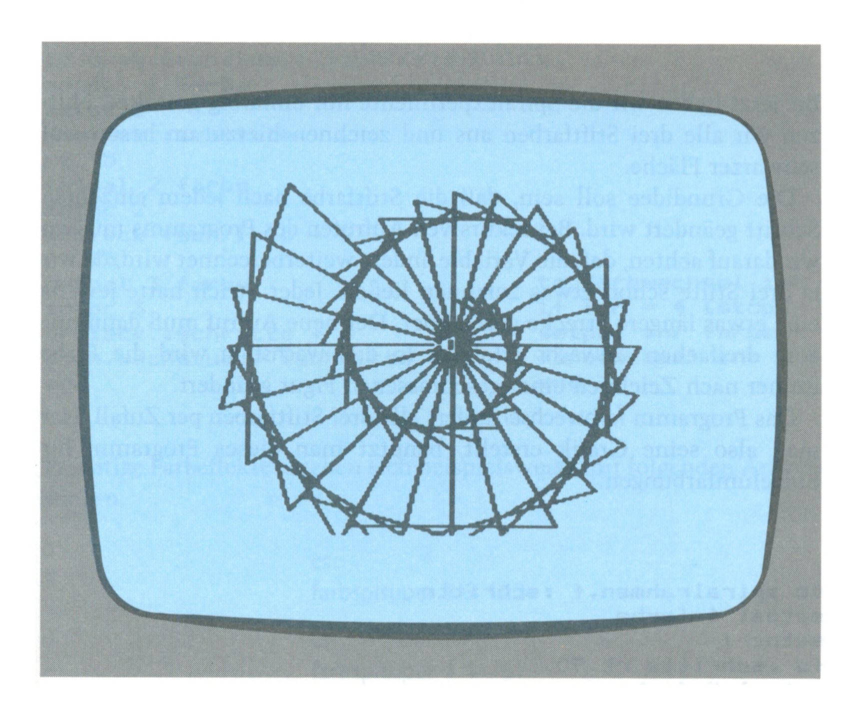


spisuper 40 0 30
spisuper 40 0 70
spisuper 40 0 89

Das nachfolgende Programm dreh.wachstum ist eine Variation unseres spidreieck. Nicht nur die Seitenlänge wächst, sondern nach Zeichnen des Dreiecks wird eine Drehung vollführt.

```
to dreh.wachstum :schritte
dreieck :schritte
rt 15
dreh.wachstum :schritte + 2
end
```

Die Zeile 2 des Programms sollte auch mit anderen bekannten Figuren (rahmen, sechseck, zwoelfeck, turm, kirche...) ausgetauscht werden. Auch die beiden Zahlenwerte «15» und «2» sollten über Variable veränderlich gemacht werden.



Bei diesen Rekursionen kann der Arbeitsspeicher des Rechners erschöpft werden. Es kommt die Meldung:

I'm out of space...

Gleichzeitig wechselt das Promptzeichen von «?» nach «!». Zur Abhilfe geben wir den Befehl ein:

recycle

Nach einigen Sekunden ist der Speicher reorganisiert und das Promptzeichen «?» erscheint wieder.

Besser noch, man löscht überflüssige Programme im Speicher oder lädt Logo neu.

Farbiges Spiraliges

Bis jetzt haben wir die Spiralexperimente nur einfarbig gestaltet. Nutzen wir alle drei Stiftfarben aus und zeichnen hierzu am besten auf schwarzer Fläche.

Die Grundidee soll sein, daß die Stiftfarbe nach jedem einzelnen Schritt geändert wird. Bei rekursiven Aufrufen des Programms müssen wir darauf achten, daß die Variable anders weiterberechnet wird, da wir ja drei Stifte schon etwas zeichnen ließen. Jeder Strich hatte jeweils eine etwas längere Strecke gezeichnet. Der neue Aufruf muß dann mit dem dreifachen :zuwachs erfolgen. In dreh.wachstum wird die Farbe immer nach Zeichnen einer geschlossenen Figur geändert.

Das Programm farbwechsel ändert alle drei Stiftfarben per Zufall. Hat man also seine Grafik erstellt, benutzt man dieses Programm für Einzelumfärbungen.

```
to spiralrahmen.f :schritte
setpal 1 farbe
setpc 1
fd :schritte rt 90
setpal 2 farbe
setpc 2
fd :schritte + 2 rt 90
setpal 3 farbe
setpc 3
fd :schritte + 4 rt 90
spiralrahmen.f :schritte + 6
end
```

```
to farbspisuper :schritte :zuwachs :grad
setpal 1 farbe
setpc 1
fd :schritte rt :grad
setpal 2 farbe
setpc 2
fd :schritte + :zuwachs rt :grad
setpal 3 farbe
setpc 3
fd :schritte + :zuwachs * 2 rt :grad
farbspisuper :schritte + :zuwachs * 3 :zuwachs :grad
end
```

```
to dreh.wachstum.f :schritte
setpal 1 farbe
setpc 1
dreieck :schritte
rt 15
setpal 2 farbe
setpc 2
dreieck :schritte + 2
rt 15
setpal 3 farbe
setpc 3
dreieck :schritte + 4
dreh.wachstum.f :schritte + 6
end
```

```
to farbwechsel :nr
if :nr = 4 [stop]
setpal :nr farbe
farbwechsel :nr + 1
end
```

Prächtige Farbeffekte ergeben sich beispielsweise mit folgenden Anweisungen:

```
cs
farbspisuper 3 3 122

cs
farbspisuper 1 1  61

cs
farbspisuper 1 1  31

cs
farbspisuper 1 1  91
```

Nach einem Abbruch mit ESC testen wir einmal das Umfärben mit den Anweisungen:

```
farbwechsel 0
repeat 10 [farbwechsel 1]
```

Man kann weitere Effekte erreichen, indem man Teile der Grafik herausfiltert. Stifte müssen nur mit dem Hintergrund farbgleich sein.

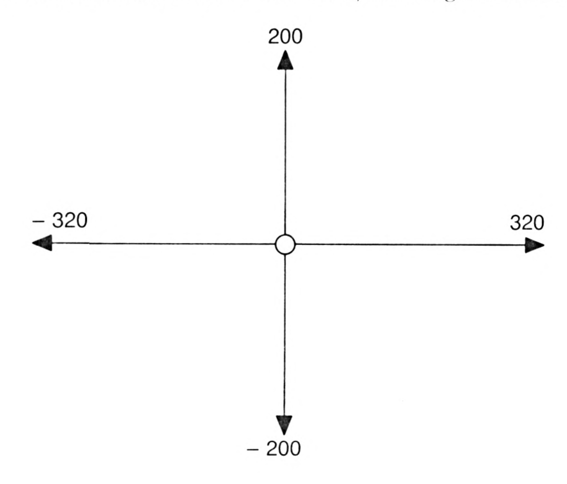
```
setpal 1 pal 0
setpal 2 pal 0
```

Die Schildkröte kriegt Zustände

In den vergangenen Lektionen haben wir eine Fülle von Möglichkeiten kennengelernt, um die Schildkröte zu unterschiedlichen Aktionen zu veranlassen. Durch entsprechende Befehle haben wir den Zustand der Schildkröte ändern können. Wir können die Schildkröte veranlassen, daß sie

- eine Spur zeichnet
- keine Spuren hinterläßt
- unsichtbar wird
- sich wieder zeigt
- Striche ausradiert und
- verschiedene Farbstifte einsetzt

Die meisten dieser Befehle zum Ändern der Zustände beginnen mit set. Nachfolgend lernen wir weitere nützliche Befehle kennen, die uns das Wählen eines neuen Standorts für die Schildkröte leichtmachen. Erinnern wir uns an unsere Programme linksunten und mitterechts in Lektion 10. Mit dem Befehl setpos ist das ganz einfach. pos ist die Abkürzung von Position. Mit setpos wird ein exakter Punkt auf dem Bildschirm vorgeschrieben, auf den die Schildkröte sofort zumarschiert. setpos hat in eckigen Klammern zwei Zahlen als Eingabe. Die beiden Zahlen beschreiben wie beim Spiel «Schiffeversenken» mittels Zeile und Spalte die jeweilige Position. Der Bildschirm bei Schneider-Logo ist ein Gitternetz oder exaktes Koordinatennetz, das folgende Bereiche hat:



Mit dem selbstdefinierten Befehl kreuz zeichnen wir dieses Kreuz nach.

```
to kreuz
setpos [0 0]
cs
fd 200
setpos [0 0] rt 90
fd 400
setpos [0 0]
end
```

Testen wir nun verschiedene Eingaben für setpos:

```
setpos [100 90]
setpos [50 30]
setpos [20 −20]
setpos [−40 40]
setpos [−100 −80]
```

Mit dem Befehl setpos [−100 −80] kommen wir in die linke untere Ecke. Erstellen wir den Befehl gehzupunkt. Damit beim Positionieren nicht der lästige Strich entsteht, wird pu und pd benutzt.

```
to gehzupunkt :ort
pu
setpos :ort
pd
end
```

Testen wir dieses Hilfsmittel und löschen vorher den Schirm.

```
cs
gehzupunkt [−100 −80]
gehzupunkt [100 80]
```

setpos kann andererseits benutzt werden, um ganz gezielt vorgegebene Punkte zu verbinden. Nehmen wir als Beispiel die Aufgabe, eine Strecke zu zeichnen. Eine Strecke hat einen Anfangspunkt und Endpunkt. Der Punkt A soll die Werte (10 20) und B (60 90) haben.

```
cs
gehzupunkt [10 20]
setpos [60 90]
```

Mit dem Befehl seth kann sofort eine beliebige Himmelsrichtung vorgegeben werden. seth ist die Abkürzung von setheading und bedeutet setze Kurs auf. Die Himmelsrichtungen werden von 0 bis 360 Grad angegeben. Norden entspricht 0 oder 360 Grad, Osten 90 Grad, Süden 180 und Westen 270 Grad.

Mit seth und gehzupunkt haben wir jetzt schöne Routinen, um die Schildkröte bei Zeichnungen und Grafikexperimenten schnell umzudirigieren und dann Figuren und Effekte zu Gesamtbildern zusammenzusetzen.

Man kann schnell mehrere Punkte verbinden, um dann Figuren zu erzeugen, die vorher auf einem Blatt Millimeterpapier entworfen worden sind.

Testen wir das Programm mit folgendem Beispiel:

```
punktverbinder [[−40 0] [−30 30] [60 −10] [0 −80]]
```

Im Vorgriff auf Kapitel 3 zeigen wir, wie solche Werte bequem in einem Programm gesammelt und als Wertmenge einem Programm eingegeben werden können.

Das Programm wird dann wie folgt aufgerufen:

```
punktverbinder bild1
```

Auf gleiche Weise können wir Figuren und Bildchen entwerfen und sie unter einem Programmnamen zusammenfassen.

Kreise und Bögen

Im Anhang sind einige nützliche Programme aufgelistet, mit denen man Kreise und Kreisbögen in verschiedener Größe zeichnet. Diese Befehle lauten:

```
kreisl ⟨radius⟩
kreisr ⟨radius⟩
bogenl ⟨radius⟩ ⟨grad⟩
bogenr ⟨radius⟩ ⟨grad⟩
```

Wir erzeugen ein einfaches Ornament durch Drehen des Kreises:

```
repeat 8 [kreisr 40 rt 45]
```

Oder erstellen wir die olympischen Ringe mit ihren Farben. Wir können hierzu gut unser Programm gehzupunkt einsetzen.

Kreisbögen können Grundelemente für verschiedene einfache Figuren werden, die zu einem Ganzen zusammengesetzt werden.

Tapeten

Nachfolgend wird ein Grundprogramm gezeigt, mit dem der gesamte Bildschirm mit Tapetenmustern gefüllt wird. Mit diesem Programm können viele Tapeten in unterschiedlichsten Farben und mit den schrägsten Mustern erzeugt werden – wir müssen uns nur ein einzelnes Elementarmuster entwerfen, das dann beliebig oft wiederholt wird. Die Variablen :rechts und :runter steuern die Tapetenmusterverteilung. Damit das einzelne Muster sich nicht mit einem anderen überschneidet, darf das Einzelmuster nicht die Fläche von :runter * :rechts überschreiten. Bauen wir genügend weitere Variable ein, entstehen schöne Effekte. Die Zahl 640/:rechts steuert die Wiederholungen je Zeile. Testen wir den einfachsten Fall:

```
tapeten 20 20
```

Die Grundprogramme sehen wie folgt aus:

```
to tapeten :runter :rechts
repeat 640 / :rechts [bild1 rechts :rechts]
runter :runter
tapeten :runter :rechts
end

to rechts :wert
pu seth 90 fd :rechts seth 0 pd
end

to runter :wert
pu seth 180 fd :wert
pd seth 360
end

to bild1
repeat 4 [fd 10 rt 90]
end

to bild2
repeat 6 [fd 20 rt 120]
end
```

Neue Logovokabeln

setpos
seth
recycle

Kursende

Das kleine Einmaleins hat hoffentlich genügend Spaß gemacht und zum Experimentieren angeregt. Viele Ideen und Möglichkeiten aus Lektion 12 werden vielleicht später einmal intensiv ausprobiert werden. Wer nicht alles durchgearbeitet hat, sollte zumindest folgendes gelernt haben und anwenden können:

- □ Definieren eigener kleiner Programmbausteine
- □ Umgang mit dem Logo-Änderungsdienst
- □ Experimentieren mit und Variieren von Anweisungen mittels CTRL-Y, dem repeat-Befehl und dem Zeileneditor
- □ eigene Teilbausteine in anderen Programmen einsetzen
- □ mit Variablen (Parametern) umgehen
- □ einfache rekursive Programme formulieren
- □ die wichtigsten Computermeckereien infolge typischer Anfängerfehler verstehen. Wer das nicht mehr genau weiß, kann im Anhang in der Logo-Kurzgrammatik nachsehen
- □ Programme auf Diskette oder Papier ausgeben

Wir wollen jetzt nicht mit dem großen Logo-Einmaleins fortfahren. Die nachfolgenden Abschnitte bieten weitere Programmbeispiele mit Wörtern, Sätzen und Tönen. Es wird nicht mehr alles in der bisherigen Ausführlichkeit dargestellt. In den Forschungsaufträgen werden neue Logovokabeln beschrieben und angewendet, was aber nicht zwingend gelesen werden muß.

Wer interessante Programmbeispiele gleich testen will, sollte es tun. Dennoch sollte man Kapitel 3 und 4 im Kern erarbeiten, da die Vokabeln op und if wesentliche Bestandteile des großen Einmaleins sind. Ohne sie kommen wir nicht weiter und bleiben bei 9×9 stehen.

2
Hallo, Herr Gutenberg – wir drucken auch

2.1 Wörter und Sätze

Logo unterscheidet zwischen einem Wort und einem Satz. Ein Satz besteht aus mehreren Wörtern, genauso wie wir es von unserer Sprache her gewohnt sind. Das Wort ist am Anführungszeichen zu erkennen. Es besteht aus zusammenhängenden Buchstaben oder anderen Zeichen. Der Satz ist an den eckigen Klammern erkenntlich. Alle Wörter eines Satzes stehen somit zwischen eckigen Klammern. Wörter und Sätze können wir drucken. Ein Satz wird in Logo auch als Liste bezeichnet.

```
?
?pr "ich
ich

?pr "ich
ich

?pr [ich gehe spazieren]
ich gehe spazieren

?pr "a
a

?pr [a]
a

?pr "

?pr []

?pr [ich [a b c] [] [du da]]
ich [a b c] [] [du da]
```

```
?pr 412
412

?pr "412
412

?pr "3+4
7

?pr 3*4
12
```

Der Sonderfall eines Wortes ist der Einzelbuchstabe. Somit besteht ein Wort aus einem oder mehreren Buchstaben oder auch aus keinem Buchstaben. Solch ein «leeres» Wort führt beim Drucken zu einer Leerzeile, weil ja in der Zeile nichts außer dem leeren Wort abgebildet wird.

Zahlen sind ebenfalls Wörter und können mit oder ohne Anführungsstriche benutzt werden. Wörter innerhalb einer Liste werden ohne Anführungsstriche angegeben.

Ein Satz oder eine Liste kann aus einem, mehreren oder keinem Listenelement (Wort) bestehen. Ein Element einer Liste kann selber eine Liste sein.

Beim Drucken werden die Anführungsstriche und die äußeren eckigen Klammern nicht mitgedruckt.

2.2 Buchstabenbilder und Texte

Genauso wie wir Grafikbefehle zu einem Programm zusammengefaßt haben, können wir Druckbefehle zu einem Programm zusammenfassen. Sehen wir uns hierzu folgende einfache Programme an, die aus Buchstaben Bilder erzeugen.

```
to pyramide
pr "\ \ \ 3
pr "\ \ 333
pr "\ 33333
pr "3333333
end
```

```
to kirche
pr "
pr "\ \ c
pr "\ c\ c
pr "c\ \ \ ccc
pr "c\ \ \ c\ \ c
pr "c\ \ \ c\ \ c
pr "c\ \ \ c\ \ c
pr "ccccccccc
end

to finger
pr []
pr [\ \ @]
pr [\ \ @]
pr [\ \ @]
pr [\ \ @]
pr [\ @@@]
pr [@@@@@\ \ @]
pr [@@@@@\ @]
pr [@@@@@@]
pr [\ @@@@]
pr [\ @@@]
end
```

Ganz wichtig sind die Leerzeichen in den Bildern. Um ein Leerzeichen als Leerzeichen ausdrucken zu lassen, müssen wir es ganz besonders kennzeichnen. Wir wissen ja, daß ein Leerzeichen ein Trennzeichen ist, an dem Logo die einzelnen Teile seiner Anweisung auseinanderhält. Soll ein Leerzeichen wie ein Buchstabe betrachtet werden, müssen wir eingeben:

\ -Taste und Leertaste

Soll ein Wort aus einem Leerzeichen bestehen, dürfen vorher natürlich die Anführungsstriche nicht vergessen werden. Und nach dem Leerzeichen darf auch das Leerzeichen als Trenner nicht vergessen werden. Das sind die tückischen Fehlerquellen bei unseren Druckprogrammen.

Ebenso können wir Texte erstellen. Sehen wir uns hierzu das nachfolgende Gedicht mit seinen Versen an. Man kann Groß- und Kleinbuchstaben verwenden, auch bei den Programmnamen. Die Logovokabeln müssen immer klein geschrieben werden.

```
to gedicht
vers1
pr []
vers2
pr []
vers3
pr []
vers4
end

to vers4
pr [es war einmal eine ziege ,]
pr [die wiegte sich gern in der wiege.]
end

to vers3
pr [so wollen wir nun hin,]
pr [dann hat alles wieder sinn.]
end

to vers2
pr [pruefe bevor du dich ewig bindest,]
pr [ob du was besseres findest.]
end

to vers1
pr [die donau ist so schoen,]
pr [drum wollen wir sie seh'n.]
end
```

Auf gleiche Weise können wir ein kleines Adreßprogramm erstellen und mittels des repeat-Befehls die Anschrift zigfach drucken lassen.

```
to adresse
pr []
pr "Gerhard\ Gerdsen
pr "Bieberer\ Weg\ 32
pr [3111 Gerdau]
pr "
end
```

```
repeat 20 [adresse]
```

2.3 Adressenhandel

Unser Adreßprogramm wollen wir variabel gestalten und entsprechende Variable für Name, Straße, Postleitzahl und Wohnort vorsehen. So können wir unterschiedliche Adressen beliebig oft ausdrucken.

```
to anschrift :name :plz :ort :strasse
pr "
pr :name
pr :strasse
(pr :plz :ort)
pr []
end
```

Das Programm wird wie folgt aufgerufen:

```
?anschrift "klaus/ maus 6070 "hausen [wegerich 12]

klaus maus
wegerich 12
6070 hausen
```

Nachfolgend werden drei Eingabefehler vorgeführt. Sitzt das kleine Logo-Einmaleins? Sind die Ursachen klar?

```
?anschrift [frau meier] 2000 "hamburg
not enough inputs to anschrift
?
?
?anschrift [frau meier] "hauptstr.10 "hamburg 2000

frau meier
2000
hauptstr.10 hamburg

?
?anschrift [frau meier] "hauptstr.10 hamburg 2000
i don't know how to hamburg
?
```

Ebenso können wir die Buchstabenbilder variabel gestalten und diese mit verschiedenen Buchstaben drucken lassen. Sehen wir uns das Beispiel kircheneu an. Wir haben eine Variable :b, die für das gewünschte Zeichen verwendet wird. :b könnte auch die Abkürzung für Baustoffen sein, und wir könnten Kirchen aus verschiedensten Materialien errichten und für Mauerwerk, Fenster und Turmkreuz Variable vorsehen.

Die Operation word hat zwei Wörter als Eingaben, die zu einem Wort zusammengefügt werden. word kann auch viele Wörter als Eingabe haben, wenn alles in runde Klammern gesetzt wird.

```
to kircheneu :b
pr "
pr word "\ \  :b
pr (word "\  :b "\  :b)
pr (word :b "\ \ \  :b :b :b)
pr (word :b "\ \ \  :b "\ \  :b)
pr (word :b "\ \ \  :b "\ \  "\  :b)
pr (word :b "\ \ \  :b "\ \ \  :b)
pr (word :b :b :b :b :b :b :b :b :b)
end
```

Testen wir einmal:

```
kircheneu "#
kircheneu "!
kircheneu "stein
```

Wer eine Kirche mit verschiedenen Baustoffen bauen will, kann auch das tun.

Vielleicht versucht einer, auch größere Bilder zu drucken? Wir müssen nur zeilenweise – am besten auf kariertem Papier – mit Schreibmaschine einen Entwurf tippen und das in entsprechende Druckanweisungen umsetzen.

2.4 Forschungsaufträge

Bisher haben wir nur Wörter und Listen ausgedruckt, sie aber nicht manipuliert. Die nachfolgenden Beispiele zeigen einige neue Logovokabeln und welche Operationen sie ausführen. Im Anhang sollten darüber hinaus weitere Operationen aus dem Abschnitt «Wörter und Listen» nachgesehen und ausprobiert werden.

Neben dem pr-Befehl gibt es noch den type-Befehl. Nach dem pr-Befehl wandert der Cursor an den Anfang der Folgezeile, bei einem Drucker wird eine Zeilenschaltung mit Druckkopfrücklauf zum Anfang durchgeführt. Beim type-Befehl bleibt der Cursor oder Druckkopf genau neben dem letzten Buchstaben stehen. Testen wir auch diese Befehle aus.

```
?
?pr first "logo
l
?
?pr first [logo]
logo
?
?pr bf "logo
ogo
?
?pr bf [logo]
?
?pr bf [a b zeh]
b zeh
?
?pr bf bf [a b zeh]
zeh
?
?pr count [a b zeh]
3
?
?pr emptyp [ich]
FALSE
?
```

```
?pr emptyp []
TRUE
?
?pr first bf bf "logo
g
?
?pr bf emptyp []
RUE
?
?pr first first first bf bf [a b [zeh]]
z
?
?pr bf first first bf bf [a b [zeh]]
eh
```

Mancher wird sich gefragt haben, warum immer der pr-Befehl als erstes kommt.

Obige Anweisungen funktionieren auch ohne einen Druckbefehl. Doch ist es sinnvoll, um darauf hinzuweisen, daß Operationen Ergebnisse liefern, die zu einer Eingabe bei einer Logovokabel oder einem Programm werden müssen. Der einfachste Fall zum Ausprobieren von Operationen ist der Druckbefehl, da man sofort das gelieferte Ergebnis sieht. Andere Logoversionen würden beim Weglassen des Druckbefehls in obigen Fällen Fehler melden.

Die oben angeführten Beispiele zeigen, daß Ergebnisse von Operationen als Eingabe bei anderen Operationen eingesetzt werden können. Wir können auf diese Weise beliebig viele Operationen verketten, sofern es sinnvoll ist.

3
Unser wichtigster Lieferant ist OUTPUT

3.1 Was liefert denn OUTPUT?

Das Wort output bedeutet ausgeben oder abliefern. In Schneider-Logo ist output abgekürzt durch op. Wir sehen uns – ohne viel Erklärungen – das nachfolgende Programm an.

```
to pronomen
op [ich du er sie es wir ihr sie]
end
```

Benutzen wir das Programm pronomen.

```
?pr pronomen
ich du er sie es wir ihr sie

?pr first pronomen
ich

?pr count pronomen
8
```

Damit der pr-Befehl druckt, muß er zwingend einen Wert als Eingabe erhalten haben. Somit hat op die Liste mit den persönlichen Fürwörtern aus dem Programm herausgeschaufelt, das heißt also, die Liste an pr abgeliefert.

Genauso können wir beispielsweise Standorte der Schildkröte sammeln (vgl. Lektion 12 und das Programm punktverbinder).

```
to werte
op [[10 40] [−20 −80] [30 60]]
end
```

Betrachten wir das nachfolgende Druckprogramm. Es druckt das dritte Element der eingegebenen Variablen aus.

```
to dr.drittes :liste
pr first bf bf :liste
end

?dr.drittes [ich du er sie es]
er
```

Das Programm dr.drittes ist keine Operation. Es liefert das ermittelte dritte Element der Liste nicht ab, sondern druckt es aus. Statt pr müssen wir die Vokabel op einsetzen, damit eine Operation entsteht und der ermittelte Wert aus dem Programm herausgeschaufelt wird. Der abgelieferte Wert kann die Eingabe für Programme und andere Logovokabeln sein.

```
to drittes :liste
op first bf bf :liste
end
```

Testen wir das neu entwickelte Programm drittes.

```
?pr drittes [ich du er sie es]
er

?pr first drittes [ich du er sie]
e
```

Anhand der Beispiele sehen wir, daß drittes eine Operation ist, die sich mit anderen Logooperationen verketten läßt.

Als nächstes definieren wir eine zweite Benutzeroperation und testen, ob sich auch selbstdefinierte Operationen miteinander verketten lassen. Erstellen wir eine Operation, die jeweils die ersten drei Elemente der Eingabe entfernt.

```
to dreiweg :objekt
op bf bf bf :objekt
end
```

```
?
?pr dreiweg [ich du er sie es]
sie es

?pr dreiweg "frankfurt
nkfurt

?pr dreiweg drittes [ich du frankfurt er]
nkfurt

?pr count dreiweg [a b c d e f g]
4

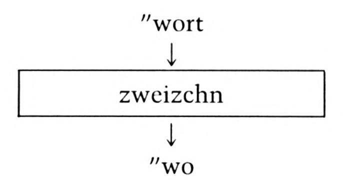
?pr bf bf dreiweg drittes [a b hahahi]
i
```

3.2 Wir schmieden neue Werkzeuge

op ist eine wichtige Logovokabel. Sie erlaubt uns, eigene Operationen zu erstellen, die sich im Verhalten gegenüber first, bf, word, random und anderen Logooperationen in nichts unterscheiden. Nachfolgend wollen wir kleine neue Werkzeuge schmieden und diese dann ausprobieren. Gerade das Verketten solcher Operationen aus einem gut gefüllten eigenen Werkzeugkasten macht die Stärke von Logo aus.

Nachfolgend sehen wir, wie kleine Werkzeuge geschmiedet werden. Wir stellen einige Operationen in Form von Aufgaben vor, wobei neben der Aufgabe die Operation schematisch mit einem Eingabebeispiel gezeigt wird. Anschließend wird jede Operation aufgelistet und getestet.

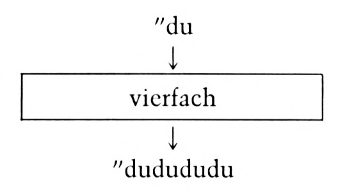
1. Schreibe eine Operation, die vom eingegebenen Wort die ersten beiden Buchstaben liefert.




```
to zweizchn :wort
op word first :wort first bf :wort
end
```

```
?pr zweizchn "wort
wo
```

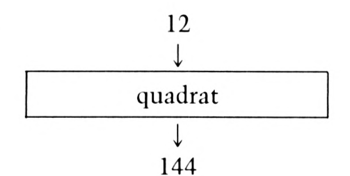
2. Schreibe eine Operation, die das eingegebene Wort vervierfacht liefert.




```
to vierfach :wort
op (word :wort :wort :wort :wort)
end

?pr vierfach "du
dudududu
```

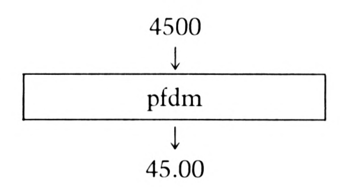
3. Schreibe eine Operation, die die eingegebene Zahl quadriert.




```
to quadrat :zahl
op :zahl * :zahl
end

?pr quadrat 12
144
```

4. Schreibe eine Operation, die einen eingegebenen Pfennigbetrag in DM umwandelt

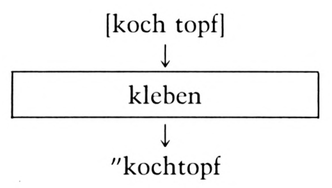



```
to pfdm :pf
op (word bl bl :pf ". last bl :pf last :pf)
end

?pr pfdm 12345
123.45
```

Bitte die Operation last aus dem Anhang «Nützliche Programmbausteine» entnehmen.

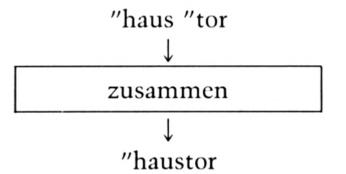
5. Schreibe eine Operation, die die beiden Wörter einer zweielementigen Liste zu einem Wort zusammensetzt.




```
to kleben :liste
op word first :liste last :liste
end

?pr kleben [koch topf]
kochtopf
```

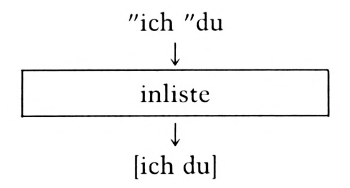
6. Schreibe eine Operation, die die zwei eingegebenen Wörter zu einem Wort zusammensetzt.




```
to zusammen :wort1 :wort2
op word :wort1 :wort2
end

?pr zusammen "haus "tor
haustor
```

7. Schreibe eine Operation, die die zwei eingegebenen Wörter in eine zweielementige Liste umwandelt.

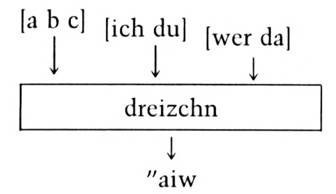



```
to inliste :wort1 :wort2
op se :wort1 :wort2
end
```

```
?pr inliste "ich "du
ich du

?pr count inliste "ich "du
2
```

8. Schreibe eine Operation, die von den drei eingegebenen Listen die ersten Buchstaben jedes ersten Wortes in der Liste zu einem Wort zusammenfügt.




```
to dreizchn :l1 :l2 :l3
op (word first first :l1 first first :l2 first first :l3)
end

?pr dreizchn [a b c] [ich du] [wer da]
aiw
```

Wie schon erwähnt, können wir natürlich die selbstdefinierten Operationen beliebig miteinander verketten. Das geht nur so lange gut, wie die Zusammensetzung sinnvoll ist und die einzelnen Operationen definitionsgemäße Eingaben erhalten. vierfach darf keine Listen als Eingabe haben. zweizchn muß mindestens ein Wort mit zwei Buchstaben als Eingabe haben usw. Sehen wir uns gleich mal einige mögliche Verkettungen an.

```
?
?
?pr quadrat quadrat quadrat 12
429981696

?pr zusammen "haus quadrat 12
haus144

?
?pr vierfach zweizchn zusammen quadrat
3 quadrat 12
91919191
```

3.3 Logo versteht deutsch

Schmieden wir nachfolgend unser eigenes und ganz persönliches Logo. Suchen wir uns bekannte und elementare Logovokabeln heraus, die wir als deutschsprachige Vokabeln benutzen möchten. Eigentlich sind alle unsere Programme deutschsprachige Erweiterungen von Logo.

Nachfolgend sind einige Grafikbefehle eingedeutscht worden. Testen wir die neuen deutschen Vokabeln anhand des Programms viereck.

```
to vor :schritte
fd :schritte
end

to zurueck :schritte
bk :schritte
end

to rechtsd :grad
rt :grad
end

to wiederhole :anz :anweisung
repeat :anz :anweisung
end

to viereck :seite
wiederhole 4 [vor 60 rechtsd 90]
end
```

Auch einige bereits bekannte Operationen werden nachfolgend eingedeutscht vorgestellt.

```
to wort :obj1 :obj2
op word :obj1 :obj2
end

to leer? :obj
op emptyp :obj
end

to zaehle :obj
op count :obj
end
```

```
to dr :was
pr :was
end

to erstes :obj
op first :obj
end

to ohneerstes :obj
op bf :obj
end
```

Leider läßt sich aus Systemgründen unser Lieferant op nicht eindeutschen. Ohne ihn geht nichts.

Zum Abschluß wollen wir die nützliche Operation item vorstellen. item heißt Element. Sehen wir, was item liefert.

```
?pr item 3 [ich du er sie es]
er

?pr item 5 "abcdef
e
```

Deutschen wir auch diese Operation ein und nennen sie n.tes.element. Denn das genau leistet item. item liefert von einem Wort oder einer Liste das n-te Element. item hat also zwei Eingaben, zuerst die Platznummer des Elements und dann das Wort oder die Liste.

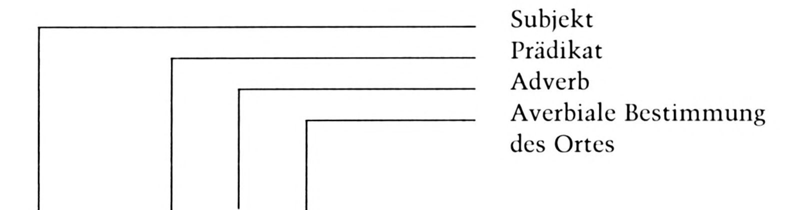
```
to n.tes.element :welches :liste
op item :welches :liste
end
```

3.4 Lange Geschichten

Ein Computer kann keine Gedichte oder Romane schreiben. Er kann aber aus einem bereitgestellten Vorrat von Satzteilen beliebige Kombinationen bilden, und das massenhaft. Solch ein Programm wollen wir erstellen. So ein Zufallsroman, der aus 60 zusammengewürfelten Zeichen besteht, sei gleich einmal vorgestellt:

```
?roman 20
detlef pfeift schnoed im kino
er lernt suess im zelt
die klasse schlaeft cool auf dem bett
ein bub weint fetzend unter der dusche
die klasse singt suess im auto
er pfeift toll auf dem bett
das stadion pfeift suess im bett
das stadion sitzt toll auf dem bett
der pauker schreibt voll im kino
der depp lacht bloed in der schule
die klasse lernt suess unter der dusche
detlef weint cool auf dem bett
gunhild pfeift nuechtern im zelt
er sitzt besoffen auf dem bett
ein bub schlaeft voll auf dem bett
detlef lacht grausam im auto
die klasse schlaeft nuechtern im bett
gunhild weint fetzend unter der dusche
er tanzt besoffen unter der dusche
fiffi weint voll im zelt
```

Ein Deutschlehrer würde mit geschultem Blick feststellen, daß alle Sätze den gleichen Aufbau haben, nämlich ein Subjekt, ein Prädikat, ein Adverb und schließlich als letztes eine adverbiale Bestimmung des Ortes.



Einigen wir uns darauf, daß der Computer jeweils einen Vorrat von zehn Subjekten, zehn Prädikaten usw. haben soll. Mit diesem Spielmaterial von 4 × 10 Satzteilen soll er dann wild drauflos «dichten». Wir erzeugen somit immer wieder einen prinzipiell gleichen Zufallssatz. Mit einer repeat-Anweisung erledigen wir das ganz bequem:

```
to roman :anzahl.zeilen
repeat :anzahl.zeilen [pr zufallssatz]
end
```

Die Operation zufallssatz stellt nach obiger Regel einen einzigen Satz zusammen:

```
to zufallssatz
op (se subjekt praedikat adverb ortsbestimmung)
end
```

Als nächstes betrachten wir die Operation subjekt, die uns jetzt aus einem Vorrat von zehn Möglichkeiten ein Subjekt liefern müßte. Die Operation n.tes.element und zufall sind uns schon bekannt. zufall liefert eine Zufallszahl zwischen 1 bis 10. Die Operation subjektliste liefert dann als Spielmaterial 10 Subjekte.

```
to subjekt
op n.tes.element zufall subjektliste
end

to subjektliste
op [er sie der\ depp der\ pauker detlef gunhild ein\ b
        das\ stadion die\ klasse fiffi]
end

to zufall
op 1 + random 10
end
```

```
?pr zufallssatz
ein bub pfeift cool im kino
```

Die restlichen Operationen praedikat, adverb und ortsbestimmung sind gleichermaßen definiert. Somit ergeben sich folgende restliche von uns definierte Funktionen, die wir jetzt auflisten.

```
to praedikat
op n.tes.element zufall praedikatliste
end
```

```
to adverb
op n.tes.element zufall adverbliste
end

to ortsbestimmung
op n.tes.element zufall ortsliste
end

to praedikatliste
op [pfeift singt lacht schlaeft sitzt geht schreibt lernt
    tanzt weint]
end

to adverbliste
op [suess grausam voll toll besoffen nuechtern cool fetzend
    bloed schnoed]
end

to ortsliste
op [auf\ dem\ bett in\ der\ schule im\ kino im\ zelt unter
    \ der\ dusche im\ auto auf\ dem\ ofen im\ unterricht
    im\ bett auf\ der\ tenne]
end
```

Bevor wir starten, noch schnell einige Tips. Nie sind Programme fehlerfrei. In Logo hat man die bequeme Möglichkeit, jede einzelne Benutzerfunktion – ob Befehl oder Operation – isoliert auszutesten. Da wir wissen, was jede Funktion leisten soll, können wir alle einzeln durchprobieren. Hier sind einige Beispiele gezeigt, wie man es machen könnte, wenn Fehlermeldungen auf eines der selbstdefinierten Programme hinweisen. Besser ist aber immer, vorher Einzeltests zu machen. Gerade die Satzteillisten sind tückisch, weil man sich mit der Anzahl verzählen kann. Es sollten ja zehn Elemente je Liste sein. Mit count läßt sich das einfach überprüfen.

```
?pr subjekt
die klasse

?pr ortsbestimmung
im unterricht

?pr count ortsliste
10
?
```

```
?pr count subjektliste
10
?pr count adverbliste
10
?
?pr n.tes.element 10 subjektliste
fiffi
?
?pr zufall pr zufall pr zufall
9
8
4
?
?pr n.tes. element zufall ortsliste
auf der tenne
```

Wenn wir halbwegs sicher sind, daß alles richtig ist, sollten wir jetzt eingeben:

```
roman 100
```

3.5 Forschungsaufträge

Schlagen wir im Anhang die Übersicht der Logovokabeln für Wörter und Listen auf. Wir finden dort viele Vokabeln, die am Ende den Buchstaben «P» haben. Der Buchstabe steht für «Prüfung». Diese Operationen sind Prüfoperationen. Geliefert wird immer "TRUE (wahr) oder "FALSE (falsch).

Machen wir uns mit ihnen vertraut, und probieren wir ein wenig herum:

```
?pr emptyp "tasse
FALSE

?pr emptyp [diese liste]
FALSE

?pr emptyp bf bf bf "tee
TRUE
```

```
?pr emptyp []
TRUE
```

Testen wir als nächstes wordp der Reihe nach mit den Eingaben "wort, [wort] und 123. Geprüft wird, ob es sich um ein Wort handelt.

Manchmal ist es nützlich, mehrere Prüfergebnisse miteinander zu verknüpfen. Wir könnten beispielsweise eine Prüfung benötigen, bei der drei Bedingungen wahr ("TRUE) sein müssen. Es könnte auch sein, daß aus mehreren Möglichkeiten mindestens eine wahr sein muß. Das Verknüpfen solcher «Wahrheiten» geschieht mit den Operationen and und or. Man kann auch ein "TRUE oder "FALSE ins genaue Gegenteil umkehren, indem einfach das Wörtchen «Nicht» vorangestellt wird in Form der Operation not. Sehen wir uns in Form von Beispielen einmal solche Verknüpfungen an:

```
?pr not wordp "123
FALSE

?pr and 3 < 4 6 > 4
TRUE

?pr or 3 < 4 4 > 6
TRUE
```

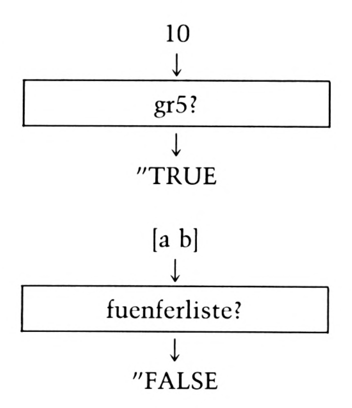
An einigen Beispielen zeigen wir eigene Prüfprogramme in der schon bekannten Darstellungsweise. Man möge sich einmal ähnliche Beispiele überlegen.

1. Erstelle ein Prüfwort, das untersucht, ob die eingegebene Zahl größer als 5 ist.

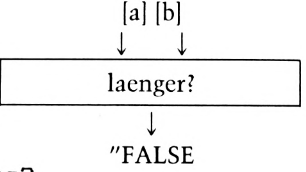
```
to gr5? :zahl
op :zahl > 5
end
```

2. Erstelle ein Prüfwort, das untersucht, ob die eingegebene Liste fünf Listenelemente hat.

```
to fuenferliste? :liste
op 5 = count :liste
end
```

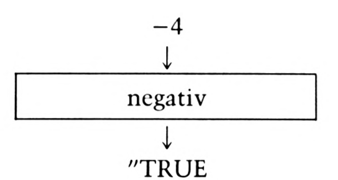


3. Erstelle ein Prüfwort, das die eingegebenen zwei Listen dahingehend untersucht, ob die erste Liste mehr Elemente als die zweite Liste enthält.




```
to laenger? :liste1 :liste2
op count :liste1 > count :liste2
end
```

4. Erstelle ein Prüfwort, das untersucht, ob die eingegebene Zahl kleiner als null ist.



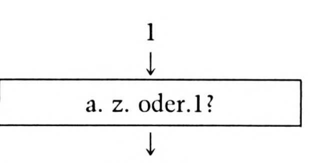

```
to negativ? :zahl
op :zahl < 0
end
```

5. Erstelle ein Prüfwort, das seine vier Eingaben untersucht, ob sie alle miteinander gleich sind.

2 2 2 1
↓ ↓ ↓ ↓
allegleich?
↓
"FALSE

```
to allegleich? :a :b :c :d
op (and :a = :b :b = :c :c = :d)
end
```

6. Erstelle eine Prüfwort, das das eingegebene Zeichen untersucht, ob es der Buchstabe "A, der Buchstabe "Z oder die Ziffer "1 ist.




```
to a.z.oder.1? :obj
op (or :obj = "a :obj = "z :obj = 1)
end
```

4
Einmal ist keinmal – Programmschleifen

4.1 Was ist, wenn ...?

In den vorangegangenen Abschnitten sind alle Anweisungszeilen schön der Reihe nach ausgeführt worden. Doch es gibt häufig Bedingungen, von denen etwas abhängig ist. Bilden wir nur einige Wenn-Sätze, und schon ist alles klar.

Wenn sie nicht gestorben sind,
dann leben sie noch heute.
Wenn im Zeugnis Fünfer sind,
dann gibt es Ärger,
sonst nicht.
Wenn meine Freundin kommt,
dann freue ich mich,
sonst suche ich mir eine neue.
Wenn der Brief bis zu 20 Gramm wiegt,
dann kostet er Normalporto,
sonst mehr.

Statt «Wenn» könnte man auch «Falls» und statt «Sonst» auch «Andernfalls» wählen. Im Englischen treten hierfür die Worte ein:

IF (Bedingung)
THEN Anweisungen für den Wahr-Fall (Ja-Zweig)
ELSE Anweisungen für den Falsch-Zweig (Nein-Fall)

Liegt nur der Wahr-Fall vor, spricht man von einer einseitigen Programmverzweigung. Im anderen Fall spricht man von einer zweiseitigen Programmverzweigung.

Schneider-Logo benutzt nur die Vokabel if. THEN und ELSE sind überflüssig. Nach der Bedingung folgt eine Anweisungsliste für den Ja-Fall oder noch eine zweite Liste für den Nein-Fall.

Das nachfolgende einfache Beispiel zeigt eine zweiseitige Verzweigung. Abhängig von der Tageszeit soll entweder mit «Guten Abend» oder «Guten Tag» gegrüßt werden.

```
to gruessen :tageszeit
if :tageszeit > 18 [pr [guten abend]] [pr [guten tag]]
end
```

```
?gruessen 19.10
guten abend
?gruessen 12
guten tag
```

Die gleiche Aufgabe hätte auch mittels einer einfachen Verzweigung gelöst werden können.

```
to gruss :tageszeit
if :tageszeit > 18 [pr [guten abend]]
if :tageszeit < 18 [pr [guten tag]]
end
```

```
?gruss 19.10
guten abend
?gruss 12
guten tag
```

In den Programmverzweigungen können auch Programme aufgerufen werden. Nehmen wir den Fall an, daß abhängig vom eingegebenen Datentyp (Liste oder Wort) entweder eine Listenverarbeitungsroutine oder eine Wortverarbeitungsroutine aufgerufen wird. Anschließend wird unabhängig vom Datentyp ein drittes Programm durchlaufen.

```
to verzweigung :objekt
if wordp :objekt [prog2 :objekt] [prog1 :objekt]
programm3
end
```

```
to prog1 :liste
pr [ich bin das listenprogramm]
(pr [und verbreite die liste] :liste)
end

to prog2 :wort
pr [ich bin das wortprogramm]
(pr [und verarbeite die worte] :wort)
end

to programm3
pr [ich bin der teil nach der verzweigung]
pr [und ich heisse programm3]
end
```

```
?verzweigung [ich du er]
ich bin das listenprogramm
und verarbeite die liste ich du er
ich bin der teil nach der verzweigung
und ich heiße programm3
```

4.2 Die Stopper

Bei Programmverzweigungen gibt es sehr schnell kleine Probleme. Sehen wir uns das Beispiel hallo einmal an. Wäre es beispielsweise nach 22 Uhr, so wäre mit der ersten Anweisung sofort das Richtige erkannt worden. Das Durchlaufen der folgenden Anweisungen wäre überflüssig.

```
to hallo :tageszeit
if or :tageszeit > 22 :tageszeit < 4 [pr [gute nacht]]
if :tageszeit > 18 [pr [guten abend]]
if :tageszeit > 12 [pr [guten tag]]
if :tageszeit > 4 [pr [morg'n]]
end
```

```
?hallo 23
gute nacht
guten abend
guten tag
morg'n
```

Helfen können wir uns einfach, indem nach Eintritt des Ja-Falles das Programm abgebrochen wird. Solch eine Abbruchbedingung finden wir in unserem Stopper, dem Befehl stop. An jeder beliebigen von uns gewünschten Stelle können wir mit stop ein Programm abbrechen.

Fügen wir in unserem Programm hallo entsprechende stop-Befehle ein und überprüfen erneut unser Programm.

```
to hallo1 :tageszeit
if or :tageszeit > 22 :tageszeit < 4 [pr [gute nacht] stop
if :tageszeit > 18 [pr [guten abend] stop]
if :tageszeit > 12 [pr [guten tag] stop]
if :tageszeit > 4 [pr [morg'n]]
end
```

Ein naher Verwandter des Stoppers ist der uns bereits bekannte Lieferant op. op stoppt auch alles. Gegenüber dem Stopper ist bei ihm nicht einfach Schluß, sondern er schaufelt noch Ergebnisse aus dem Programm heraus und liefert sie für mögliche Weiterverwendungen ab. Sehen wir uns das entsprechend gestaltete Grußprogramm an.

```
to hallo3 :tageszeit
if or :tageszeit > 22 :tageszeit < 4 [op [gute nacht]]
if :tageszeit > 18 [op [guten abend]]
if :tageszeit > 12 [op [guten tag]]
if :tageszeit > 4 [op [morg'n]]
end
```

4.3 Der große Dreh

Beim Lernen des kleinen Logo-Einmaleins haben wir eine Fülle von rekursiven Programmen benutzt. Diese Kreisläufer haben wir nur gewaltsam über die Tastatur zum Abbruch zwingen können. Statt des wenig eleganten Ziehens der Notbremse (ESC-Taste) wollen wir jetzt mit Stil Rekursionen zu einem kontrollierten Ende bringen.

Sehen wir uns hierzu noch einmal das Programm spisuper aus Lektion 12 an.

Wir müssen, um kontrolliert abzubrechen, nur eine Anzahl vorsehen, die die Wiederholung vorschreibt oder einen Vergleichswert vorgibt.

Wir können nun genau diesen Grenzfall abfragen. Tritt er ein, verrichtet Freund Stopper seinen Job.

```
to spisuper :schritte :zuwachs :grad :ende
if :schritte > :ende [stop]
fd :schritte rt :grad
spisuper :schritte + :zuwachs :zuwachs :grad :ende
end

to spisuper1 :schritte :zuwachs :grad :anzahl
if :anzahl = 0 [stop]
fd :schritte rt :grad
spisuper1 :schritte + :zuwachs :zuwachs :grad :anzahl - 1
end
```

Denken wir auch an unser Programm punktverbinder in Lektion 12. Das Zeichen einer Umrißlinie geht kontrolliert nach dem letzten Zahlenpaar zu Ende.

```
to verbinden :werte
if emptyp :werte [stop]
setpos first :werte
verbinden bf :werte
end
```

Nun wollen wir einige weitere Beispiele vorstellen, die sich mit der Verarbeitung von Wörtern und Listen befassen. Das nachfolgende Druckprogramm unter druckt die Buchstaben eines Wortes oder die Elemente einer Liste untereinander aus.

```
to unter :obj
if emptyp :obj [stop]
pr first :obj
unter bf :obj
end
```

Das anschließende Programm schreibt die eingegebenen Wörter rückwärts. Was passiert eigentlich, wenn der type-Befehl gegen den pr-Befehl ausgetauscht wird?

Die Operation last ermittelt das letzte Element eines Worts oder Satzes. last sollte aus dem Anhang «Nützliche Programme» entnommen werden.

```
to spiegel :was
if emptyp :was [stop]
type last :was
spiegel bl :was
end
```

Variieren wir unser Programm spiegel und erstellen das Programm spiegelneu. Es soll nur Listen verarbeiten und zusätzlich das einzelne Wort auch umgekehrt schreiben. Beispielsweise wird die Liste [ich du er] als [re ud hci] ausgegeben werden.

```
to spiegelneu :liste
if emptyp :liste [stop]
spiegel last :liste
type "\
spiegelneu bl :liste
end
```

Ermitteln wir die Quersumme von Zahlen. Die Ziffern werden einzeln herausgefischt und in einem Speicher aufaddiert. Richten wir als Trick in der Kopfzeile hierfür gleich die entsprechende Variable ein.

```
to quersumme :zahl :summe
if emptyp :zahl [pr :summe stop]
quersumme bf :zahl :summe + first :zahl
end
```

Abschließend wollen wir Zeugnisse schreiben.

Für bestimmte Fächer sollen die entsprechenden Noten nicht als Ziffer, sondern als Text ausgegeben werden. Nehmen wir die übliche Notenskala mit den Noten von sehr gut bis ungenügend.

Das Programm minizeugnis soll Noten ausdrucken, die als Zahlen in einer Liste vorgegeben werden. Die zweite Zeile ist uns schon recht vertraut. ntes haben wir in Abschnitt 3.4 in der Form ntes.element kennengelernt, doch hier ist ntes keine Operation, sondern letztlich ein Druckbefehl. notenskala definiert uns die Liste mit Daten.

```
to minizeugnis :noten
if emptyp :noten [stop]
ntes notenskala first :noten
minizeugnis bf :noten
end

to notenskala
op [sehr\ gut gut befriedigend ausreichend mangelhaft
    ungenuegend]
end
```

```
to ntes :obj :stelle
if :stelle = 1 [pr first :obj stop]
ntes bf :obj :stelle - 1
end
```

?minizeugnis [6 1 5 2 3 4 2]
ungenuegend
sehr gut
mangelhaft
gut
befriedigend
ausreichend
gut

Wir brauchten minizeugnis nur um die Eingabe :faecher zu erweitern und in der zweiten Programmzeile zusätzlich das erste Element von :faecher drucken lassen, dann wäre unser Zeugnisprogramm erledigt:

```
to zeugnis :noten :faecher
if emptyp :noten [stop]
type first :faecher
ntes notenskala first :noten
zeugnis bf :noten bf :faecher
end
```

?zeugnis [6 1 2] [bio deutsch mathe]
bioungenuegend
deutschsehr gut
mathegut

Der Test zeigt, daß ein kleiner Schönheitsfehler vorliegt. Die Noten müßten weiter rechts stehen, und die Anfangsbuchstaben sollen genau untereinander stehen. Wir müßten eine Schreibstelle vorgeben, von der an in jeder Zeile das Fach geschrieben werden soll. Solch eine Tabulatorfunktion erstellen wir, indem wir das Fach mit konstanter Länge ausgeben. (Schneider-Logo hat keine Cursor-Steuerung, um eine Tabulatorroutine zu erstellen). Jedes Fach wird linksbündig mit angehängten Füllzeichen ausgegeben.

```
to lbuend :wort :laenge :zchn
type :wort
repeat (:laenge - count :wort) [type :zchn]
end
```

Die Variable :laenge ist die gewünschte Gesamtlänge. :zchn gibt das gewünschte Füllzeichen an.

Fügen wir jetzt lbuend in unserem Programm ein, und testen wir es:

```
to zeugnis :noten :faecher
if emptyp :noten [stop]
lbuend first :faecher 15 ".
ntes notenskala first :noten
zeugnis bf :noten bf :faecher
end
```

```
?zeugnis [6 1 2] [bio deutsch mathe]
bio      ungenuegend
deutsch  sehr gut
mathe    gut
```

4.4 Forschungsaufträge

Bisher haben unsere Rekursionen nur gedruckt. Wie wäre es, wenn wir diese rekursiven Programme nicht von unserem Freund Stopper, sondern von seinem Verwandten, dem Lieferanten op, abbrechen lassen. Das hätte auch noch den Vorteil, daß wir ein Ergebnis geliefert bekommen, mit dem wir weiterarbeiten könnten. Simples Ausdrucken ist eigentlich nur ein Sonderfall. Nehmen wir uns das Programm spiegel einmal vor und überlegen, wie es mit op gestaltet werden müßte.

Das neue Programm spiegel soll also das Gespiegelte als Ganzes abliefern. Wird ein Wort eingegeben, soll op ein ganzes Wort liefern. Stellen wir das Programm in seinem ersten Ausbauschritt der alten stop-Version gegenüber.

```
to spiegel :obj              to spiegel :wort
if emptyp :obj [stop]        if emptyp :wort [op :wort]
type last :obj               ??????
spiegel bl :obj              end
end
```

Bis hierher ist eigentlich nichts anderes zu erwarten. stop mußte rausfliegen, da ja mit op abgebrochen werden soll. Auch der Druckbefehl mußte entfallen. Was kommt statt dessen?

Übrigens läuft das neue Spiegelprogramm bereits in der abgebildeten Form, doch nur für den Sonderfall eines leeren Wortes.

```
?pr spiegel"

?
```

Geliefert wird also das leere Wort, das gedruckt wird. Durch den pr-Befehl wird dann auf die nachfolgende Zeile gesprungen, und das Promptzeichen erscheint am Zeilenanfang wie gehabt.

Was muß also weiter geschehen? Klar ist auch, daß wir als Bestandteil der restlichen Anweisung spiegel erneut aufrufen werden. Die Eingabe ist dann bl :wort.

Das Ergebnis muß irgendwann mit op abgeliefert werden.

```
to spiegel :wort
if emptyp :wort [op :wort]
op ???? spiegel bl :wort
end
```

Jetzt kommt das eigentlich Schwierige. Wir wollen als Ergebnis alle Buchstaben nicht wie bisher einzeln ausdrucken, sondern sammeln, das heißt, sie zu einem Wort zusammenfügen. Und das leistet die Logovokabel word.

```
to spiegel :wort
if emptyp :wort [op :wort]
op word last :wort spiegel bl :wort
end
```

Testen wir das neue spiegel mit einem Buchstaben als Eingabe.

```
?pr spiegel "a
a
?
```

Denken wir den Fall genau durch. Die Stopperzeile wird nicht ausgeführt, weil das eingegebene Wort ja nicht leer ist. Also kommt die zweite Zeile zur Ausführung. Der letzte Buchstabe des Worts wird geangelt und zur ersten Eingabe von word. Mit dem Rest des Worts wird spiegel erneut aufgerufen. Dieser erneute spiegel-Aufruf liefert jetzt das leere Wort, das die zweite Eingabe von word wird.

Jetzt probieren wir spiegel mit zwei Buchstaben als Eingabe aus:

```
?pr spiegel "ha
ah
?
```

Jeder sollte versuchen, genau systematisch und Schritt für Schritt die Sache durchzuspielen. Rekursionen wirken zwar einfach, sind es aber nicht, solange man es nicht voll verstanden hat. Das dauert schon eine gewisse Zeit – und niemand sollte sich grämen, wenn er nicht gleich Experte für Rekursionen mit op ist.

Nachfolgend werden noch einige bekannte Beispiele und zwei neue gezeigt, mit denen man sich mal beschäftigen kann, wenn das Bisherige einem nicht schon gereicht hat.

```
to spiegel :was
if emptyp :was [op :was]
op word last :was spiegel bl :was
end

to spiegelneu :liste
if emptyp :liste [op []]
op se spiegel last :liste spiegelneu bl :liste
end

to quersumme :zahl
if emptyp :zahl [op 0]
op first :zahl + quersumme bf :zahl
end

to nzchn :n :zchn
if :n = o [op "]
op word :zchn nzchn :n - 1 :zchn
end

to rechtsbuendig :laenge :wort :zchn
op word nzchn :laenge - count :wort :zchn :wort
end
```

```
to igittigitt :wort
if emptyp :wort [op :wort]
if member? first :wort [a e i o u]
    [op word "i igittigitt bf :wort]
op word first :wort igittigitt bf :wort
end
```

Die Optionen memberp und last findet man im Anhang bei den «Nützlichen Programmbausteinen».

5
Ich gebe Dir den Namen...

5.1 Jedes Ding hat einen Namen

Manchmal besteht die Aufgabe, irgendeine Zahl, ein Wort, eine Liste oder eine Bildschirmposition aufzubewahren. Brieftauben haben die geografischen Koordinaten ihres heimatlichen Taubenschlages fest gespeichert. Auch die Schildkröte hat ihren entsprechenden Wert im Kopf, wenn es cs heißt. Angenommen, wir wollten eine bestimmte Bildschirmstelle ansteuern, so haben wir uns bisher wie folgt helfen können.

```
to stelle
op [10 80]
end
```

Mit dem Befehl setpos stelle kommen wir immer an diese eine Stelle.

Logo kennt eine weitere Möglichkeit. Wir können die Koordinaten [10 20] taufen und ihnen einen Namen geben. Nach dem Taufen können wir jederzeit dieses Zahlenpaar unter dem Taufnamen ansprechen.

Getauft wird in Logo mit dem Befehl make, beispielsweise:

```
make "stelle [10 20]
```

make hat also zwei Eingaben. Nach make kommt als allererstes der Taufname, der ein Wort sein muß und daher zwingend mit Anführungsstrichen beginnt. Nach dem Taufnamen folgt das Ding, ein Wort, eine Zahl oder eine Liste.

make ⟨Taufname⟩ ⟨Objekt⟩

Der make-Befehl weist einem Wort eine bestimmte Sache zu. Das Wort "stelle wird auch als Variablenname und die Sache als der Variableninhalt bezeichnet. Weisen wir in weiteren Beispielen weiteren Wörtern bestimmte Inhalte zu.

```
make "heute "dienstag
make "gestern "montag
make "freunde [gerd fritz klaus]
make "durchschnitt 2.2
make "stelle [10 20]
```

Wenn wir die Inhalte solcherart entstandener Variablen erhalten wollen, müssen wir nur statt der Anführungsstriche einen Doppelpunkt setzen.

```
?pr :heute
dienstag
?pr :gestern
montag
?pr :freunde
gerd fritz klaus
?pr :durchschnitt
2.2
```

Diese Form ist uns geläufig. Wir hätten auch sagen können, drucke die Sache (thing) des Worts "heute. Schneider-Logo kennt nicht die Vokabel thing, wie andere Logoversionen. Doch sie ist sehr nützlich und ist unter dem Kapitel «Nützliche Programmbausteine» zu finden.

```
?pr thing "heute
dienstag
```

In unserem Eingangsbeispiel zum Positionieren des Cursors besteht jetzt auch folgende Möglichkeit:

```
setpos thing "stelle
setpos :stelle
```

Also aufgepaßt, daß wir immer genau wissen, was wir wollen, oder Logo wird fragen, was gemeint ist (Fehler!). Wir müssen uns nur drei Dinge merken und was sie bedeuten:

- Anführungsstriche am Anfang
- Doppelpunkte am Anfang
- Leerzeichen am Anfang (weder " noch :)

Zum Löschen von Wertzuweisungen gibt es den Befehl ern (erase name [s]), mit dem man eine oder mehrere Wertzuweisungen löschen kann. Soll nur eine Wertzuweisung gelöscht werden, muß der Taufname eingegeben werden. Bei mehreren zu löschenden Variablen kann man die Namen in einer Liste angeben.

```
ern "a
ern [wort a e d]
```

Da man häufig diese Namen nicht kennt, leisten die Routinen pons und erns gute Hilfe (vgl. «Nützliche Programmbausteine» im Anhang).

5.2 Kreisverkehr

An dieser Stelle lernen wir eine weitere Möglichkeit kennen, Programmschleifen mittels des repeat-Befehls zu erstellen, was bisher nicht möglich gewesen ist. Daran anschließend werden Programmschleifen mittels des Sprungbefehls go vorgestellt.

Sehen wir uns noch einmal das Programm unter aus Abschnitt 4.4 an. Die nachfolgende Version benutzt den repeat-Befehl. Die Anzahl der Wiederholungen muß jetzt aber berechnet werden. Wir können nicht mehr abfragen, ob das Objekt leer ist. Die Berechnung der Anzahl erfolgt einfach durch Zählen der Anzahl der Elemente des Objekts.

```
to unter :wort
repeat count :wort [pr first :wort make "wort bf :wort]
end
```

Auch unser Buchstabendrehprogramm spiegel läßt sich auf diese Weise formulieren.

```
to spiegel :obj
repeat count :obj [type last :obj make "obj bl :obj]
end
```

Das Programm quersumme hat damals bestimmt Verwunderung ausgelöst, als eine zweite Programmvariable für das Ergebnis vorgesehen werden mußte. In der neuen Version wird die Variable namens "sum definiert. Sie erhält als Anfangswert die Zahl 0 zugewiesen. Als letztes schaufeln wir mit op die Summe :sum heraus, so daß das Teilprogramm zu einer Operation wird und als Lieferant für die Eingaben anderer Programme werden kann.

```
to quersumme :zahl
make "sum 0
repeat count :zahl [make "sum :sum + first :zahl
                    make "zahl bf :zahl]
op :sum
end
```

Das Spiegelprogramm können wir ebenso als Operation definieren.

```
to spiegel1 :obj
make "erg "
repeat count :obj [make "erg word :erg last :obj
                   make "obj bl :obj]
op :erg
end
```

Bei allen vorangegangenen Beispielen erkennt man, daß die Anweisungsliste schnell zu einem unangenehmen langen Bandwurm werden kann und unleserlich wird. Helfen wir uns ganz einfach, indem wir für den Listeninhalt ein zusätzliches Kernprogramm definieren.

```
to spiegel1 :obj
make "erg "
repeat count :obj [spiegelkern]
op :erg
end

to spiegelkern
make "erg word :erg last :obj
make "obj bl :obj
end
```

Der Befehl go erlaubt uns, innerhalb eines Programms mehrere Anweisungszeilen zu überspringen. Die Einsprungstelle muß auf besondere Weise markiert werden. Sehen wir uns ein einfaches Beispiel an, das das Wort "schleife in einer Endlosschleife unentwegt ausdruckt. Wir springen bei Programmende wieder auf den Programmanfang.

```
to schleife
label "beginn
pr "schleife
go "beginn
end
```

Mit label, gefolgt von einem Wort, wird in Programmen die Einsprungstelle markiert. Der go-Befehl verlangt als Eingabe den Namen der Einsprungstelle.

Es folgen drei bekannte Beispiele, die mittels des Sprungbefehls erstellt worden sind.

```
to unter :objekt
label "anfang
if emptyp :objekt [stop]
pr first :objekt
make "objekt bf :objekt
go "anfang
end

to quersumme :zahl
make "summe 0
label "start
if emptyp :zahl [op :summe]
make "summe :summe + first :zahl
make "zahl bf :zahl
go "start
end

to spiegel1 :obj
make "erg "
label "start
if emptyp :obj [op :erg]
make "erg word :erg last :obj
make "obj bl :obj
go "start
end
```

Nachfolgend wollen wir ein Programm erstellen, das beliebig lange Zeichenketten einer Buchstabenanalyse unterzieht. Das Programm soll ermitteln, welche Buchstaben wie oft vorkommen.

Unser Leitprogramm heißt zchn.analyse. Von ihm aus werden weitere Routinen aufgerufen.

Wir müssen uns die vorkommenen Zeichen in einer Liste merken, damit wir später diese Variablen ansprechen und ausgeben können. Bei jedem zu verarbeitenden Buchstaben wird überprüft, ob dieser Buchstabe in unserer Merkliste vorkommt. Im Nein-Fall wird dieser Buchstabe unserer Merkliste hinzugefügt. Sehen wir uns dieses Merkprogramm merke und seine Einbettung in analyse an.

Benutzen wir unsere Merkliste, um die einzelnen vorkommenden Buchstaben mit ihren Zahlenwerten gezielt auszudrucken. In einem Zusatzprogramm ausgabe wird dies geleistet. Vorher wird noch eine kleine Überschrift ausgegeben. Zur Abwechslung ist ausgabe rekursiv formuliert.

```
to zchn.analyse :wort
make "liste []
repeat count :wort [analyse]
ueberschrift
ausgabe :liste
ern :liste
end

to analyse
merke first :wort
make first :wort 1 + thing first :wort
make "wort bf :wort
end

to merke :zchn
if memberp :zchn :liste [stop]
make "liste se :liste :zchn
make first :wort 0
end

to ueberschrift
pr [bei der analyse kommen folgende buchstaben vor:]
pr []
end
```

```
to ausgabe :liste
if emptyp :liste [stop]
pr se first :liste thing first :liste
ausgabe bf :liste
end
```

Die Operationen thing und memberp sind dem Anhang zu entnehmen.

Testen wir unser Programm:

```
?zchn.analyse "hihihahahuhu
bei der analyse kommen folgende buchstaben vor:

h 6
i 2
a 2
u 2
```

Die letzte Anweisung im Leitprogramm ist wichtig, da die erzeugten Wertzuweisungen Globalvariable sind. Diese Globalvariablen würden andernfalls ständig Platz im Arbeitsspeicher belegen und sogar unerwünschte Seiteneffekte bewirken.

Programmparamter (Eingabevariable in den Kopfzeilen) sind immer Lokalvariable. Solch ein Parameter lebt nur so lange, wie der jeweilige Programmbaustein existiert. Man spricht daher auch von sogenannten lokalen Variablen, da sie nur im Geltungsbereich des jeweiligen Teilprogramms gelten.

Daher brauchen wir uns bei Namensvergaben für Parameter um irgendwelche Namensgleichheiten nicht zu kümmern. Somit könnten wir alle Teilbausteine im Prinzip mit gleichen Parameternamen versehen, ohne daß es Schwierigkeiten gibt. Parameter können wir innerhalb eines Programms beliebig mit make umdefinieren. Mit den anderen Variablen ist es anders. Sie stopfen irgendwann einmal den Arbeitsspeicher des Computers zu. Diese Wertzuweisungen existieren, solange der Computer unter Strom steht oder wir diese Wertzuweisungen löschen. Alle Teilprogramme können auf solche allgemeingültigen Wertzuweisungen zugreifen. Daher spricht man auch bei diesen Variablen von Globalvariablen.

5.3 Forschungsaufträge

Zum Abschluß wollen wir einmal einen Vergleich von rekursiven Schleifen und entsprechenden repeat-Schleifen vornehmen. Benutzen wir hierzu das Programm summe in zwei entsprechenden Varianten. Jedes Programm soll abhängig von der Variablen :zahl entsprechend oft die Ziffer «1» addieren.

```
to summe :anz
make "erg 0
repeat :anz [make "erg :erg + 1]
op :erg
end

to summe1 :anz
if :anz = 0 [op 0]
op 1 + summe1 :anz - 1
end
```

```
?pr summe1 50
150
?pr summe 50
150
?
?
?pr summe1 200
im out of space in summe1
?
?pr summe 200
200
```

Im ersten Fall messen wir die Zeit, die jedes Programm für die Addition von 50 Einsen benötigt. summe1 benötigt deutlich mehr Zeit.

Im zweiten Fall sehen wir einen wesentlichen Nachteil einer Rekursion, wenn sehr viele Wiederholungen vorgenommen werden. Die interne Buchführung zur Abwicklung von Rekursionen ist speicheraufwendig, und der Computer kann «zusammenbrechen», weil nicht genügend Speicherplatz vorhanden ist (out of space).

Wer Rekursionen nicht mag, kann sich jetzt freuen. Andere werden die elegante Programmiertechnik der Rekursion schätzen, aber nur gezielt einsetzen, um Laufzeitnachteile zu vermeiden. repeat-Schleifen sind in der Regel schneller, aber umständlicher zu formulieren.

6
Hast Du Töne...

6.1 Der sound-Befehl

In diesem Abschnitt wollen wir mit Musik, Tönen und Geräuschen experimentieren. Der wichtigste Befehl ist der zum Erzeugen von Tönen. Schneider-Logo hat einen Tongenerator, der Töne in beliebiger Höhe und Dauer erzeugt. Es liegt an uns, ob es kracht, krächzt, kratzt, knarrt, fiept, fienst, pfeift oder lieblich klingt. Mit dem Befehl sound und seinen nachfolgenden zwei Eingabewerten werden Töne erzeugt. Es lassen sich bis zu drei Töne gleichzeitig erzeugen.

Der sound-Befehl sieht formal wie folgt aus:

sound ⟨Kanalstatus⟩ ⟨Tonperiode⟩ ⟨Dauer⟩

Neben diesen drei Parametern sind noch vier weitere wahlweise möglich. Alle Parameter müssen eine Liste bilden (vgl. Benutzerhandbuch »DDI-1«).

Listen wir ein paar Beispiele auf:

```
sound [1 1000 60]
sound [1  500 30]
sound [1  100 60]
sound [1   10 60]
```

Um den Tonumfang zu erkennen, benutzen wir am einfachsten unser Programm periode in Verbindung mit dem repeat-Befehl. Mit dem pr-Befehl drucken wir jeweils die Tonperiode des erzeugten Tons aus, um schnell ein Gefühl für die Tonhöhe zu gewinnen. Die Tonperiode kann eine Ganzzahl im Bereich von 0 bis 4095 sein.

```
to periode :periode
sound (se 1 :periode 60)
pr :periode
wait 30
end
```

Rufen wir das Programm mehrfach auf:

```
repeat 100 [periode random 4095]
```
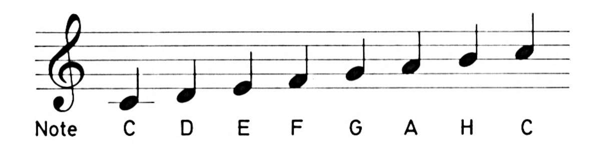
Testen wir gleich einmal, welche Frequenzen wir gerade noch wahrnehmen können.

Um den sound-Befehl bequemer benutzen zu können, definieren wir den Befehl ton:

```
to ton :kanal :freq :dauer
sound (se :kanal :freq :dauer)
end
```
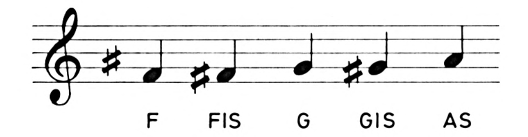
6.2 Sing ein Lied

Wer Melodien und Musikstückchen mit Schneider-Logo komponieren oder vorgegebene Noten umsetzen will, muß leider Noten vom Blatt ablesen können. Auch die Stufen der Tonleiter und ihre Bezeichnung und deren Frequenz benötigen wir. Machen wir daher einen Notenschnellkurs. Sehen wir uns als nächstes einfach die C-Dur-Tonleiter an. Der schwarze Kreis bedeutet eine viertel Note, diese Tonleiter entspricht der 2. Oktave.
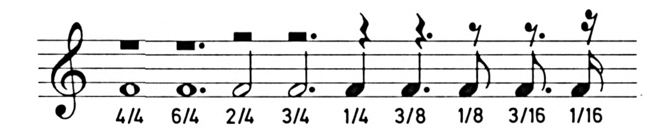


Zwischen den Noten C, D, F, G, A und H befinden sich noch Zwischennoten. Diese Noten werden mit CIS, DIS, FIS, GIS und AS bezeichnet. Sind diese Noten erwünscht, so finden wir jeweils ein Kreuz (#) der

Note vorangestellt. Bei Musikstücken, die ausschließlich bestimmte Zwischennoten verwenden, werden diese Kreuze nicht ständig vor jede Note gesetzt. Nur einmal zu Beginn der Zeile wird dieses Zeichen neben dem Notenschlüssel angeführt.



Nachfolgend werden weitere Noten mit ihrer Dauer und den zeitlich entsprechenden Pausen in ihrer Notation gezeigt.



In der folgenden Tabelle werden die Noten der ersten drei Oktaven mit ihren zugehörigen Tonperioden gezeigt. Die Werte sind dem Anhang VII: Noten und Tonperioden des Schneider-BASIC-Handbuchs entnommen.

Note	1. Oktave	2. Oktave	3. Oktave
C	956	478	239
CIS	902	451	225
D	851	426	213
DIS	804	402	201
E	758	379	190
F	716	358	179
FIS	676	338	169
G	638	319	159
GIS	602	301	150
A	568	284	142
AS	536	268	134
H	506	253	127

Für unsere Musikbeispiele sollten wir für jeden Ton ein kleines Programm schreiben, das mit op jeweils genau die zugehörige Tonperiode liefert. Die folgenden Töne sind auf diese Weise realisiert. Die Töne der dritten Oktave sind mit einem «'» gekennzeichnet und Töne der 1. Oktave mit einem nachgestellten «.».

```
to a.
op 568
end

to h.
op 494
end

to c
op 478
end

to cis
op 451
end

to d
op 426
end

to dis
op 402
end

to e
op 379
end

to f
op 358
end

to fis
op 338
end

to g
op 319
end

to gis
op 301
end

to a
op 284
end

to as
op 286
end

to h
op 253
end

to c'
op 239
end

to cis'
op 225
end

to d'
op 213
end

to dis'
op 201
end

to e'
op 190
end

to f'
op 179
end

to fis'
op 169
end

to g'
op 159
end

to a'
op 142
end
```

Mit diesen Noten können wir jetzt schnell beliebige Melodien komponieren. Hören wir uns mittels des Programms tonfolge einmal die C-Dur-Tonleiter an. Das kommt uns schon bekannt vor!

```
to tonfolge :noten
if emptyp :noten [stop]
ton 1 run (se first :noten) 60
tonfolge bf :noten
end
```

?tonfolge [c d e f g a h c']

Dieses rekursive Beispiel können wir auch mittels des repeat-Befehls formulieren. Es hat den Vorteil, etwas schneller in der Ausführung zu sein, und belegt nicht so viel Speicher wie die Rekursion.

```
to toene :noten
repeat count :noten [ton 1 run (se first :noten) 60
                              make "noten bf :noten]
end
```

Im Programm erscheint die neue Logovokabel run. run heißt «lauf» oder «mach es». run verlangt als Eingabe eine Liste. Diese Liste enthält Anweisungen, die ausgeführt werden sollen.

```
?pr [pr "hallo/ ihr/ da!!!]
pr "hallo ihr da!!!
?
?run [pr "hallo/ ihr/ da!!!]
hallo ihr da!!!
```

An dem Beispiel wird der Unterschied in der Behandlung von einer eingegebenen Liste bei pr und run deutlich. Der Ausdruck se first :noten [] liefert in unserem Beispiel den Wert [c]. Diese Liste mit dem Buchstaben «c» als Inhalt wird ausgeführt (evaluiert) und liefert die von uns gewünschte Tonperiode 478 als Wert für den zweiten Parameter von sound.

Die run-Vokabel ist nicht einfach. Wer es genau wissen will, sollte im Programm die Zeile wie folgt ändern:

```
ton 1 first :noten 60
```

Lassen wir jetzt das Programm erneut ablaufen, so erscheint die Fehlermeldung «ton doesn't like c as input in tonfolge». Deshalb muß also c mit
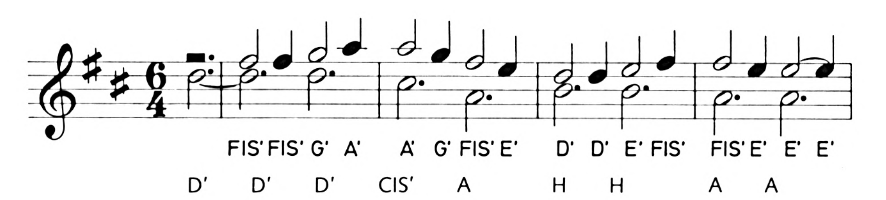
run erst ausgewertet werden. Wem die run-Konstruktion nicht gefällt, sollte es bei der geänderten Zeile belassen. Als Eingabe für :noten müssen wir dann Listen vorsehen, die Zahlen enthalten. Dieser Weg ist bei der Umsetzung von Noten aber beschwerlicher.

Entsprechend könnten wir natürlich auch den zweiten Tongenerator für eine zweite Stimme aktivieren. Demonstrieren wir die zweite Stimme, indem wir einige Akkorde bilden.

```
to zweistimmig :noten :noten2
if emptyp :noten [stop]
ton 1 run (se first :noten) 60
ton 2 run (se first :noten2) 60
zweistimmig bf :noten bf :noten2
end
```

zweistimmig [c e g] [c' e' g']

Nehmen wir als Aufgabe, das Leitmotiv aus Beethovens Neunter Symphonie umzusetzen.



In den oben abgebildeten vier Takten sind die Notenbezeichnungen der ersten und zweiten Stimme übereinandergeschrieben. In diesem Stück kommen nur dreiviertel, halbe und viertel Noten vor. Wir müssen die Dauer der Töne steuern.

Nehmen wir die kürzeste Notendauer als Grundlage und stellen die anderen Noten als das entsprechend Mehrfache dar. Also werden aus einer dreiviertel Note in unserem Schema drei einzelne Viertelnoten. Das ist schon eine kleine Abweichung, doch so kommen wir schnell zu unserem Musikwunsch.

Die Melodien ergeben sich damit wie folgt:

```
to m1
op [p p p fis' fis' g' g' a' a' a' g' fis' fis' e'
    d' d' d' e' e' fis' fis' fis' e' e' e' e']
end

to m2
op [d' d' d' d' d' d' d' d' d' cis' cis' cis' a a a
    h h h h h h a a a a a a]
end

to p
op 0
end
```

Da in der zweiten Stimme eine Pause vorgesehen ist, haben wir die Note p (für Pause) vorgesehen. p erzeugt einen unhörbaren Ton. Rufen wir gleich unseren Beethoven auf.

zweistimmig m1 m2

Wer genau hingehört hat, stellt fest, daß die Töne nicht gleichzeitig einsetzen. Zum Schluß bleibt ein Ton etwas länger hörbar. Beide Stimmen müßten synchronisiert werden. Solch einen Gleichlauf erreicht man, indem man den jeweils ersten Ton jeden Kanals erzeugt, aber nicht ausführen läßt, ihn stoppt. Hieran anschließend gibt man einen Freigabebefehl für beide Kanäle gleichzeitig. In unserem Beispiel release 3, da wir die Kanäle A und B benutzen. Einzelheiten sollten im CPC-464-Handbuch nachgesehen werden. Gleiches gilt für die Rendez-vous-Technik (siehe die zweistelligen Nummern für den Kanalstatus von A und B). Testen wir nachfolgendes Programm melodie1:

```
to melodie1 :noten :noten2 :dauer
anfang
release 3
label "anf
if emptyp :noten [stop]
ton 17 run (se first :noten) :dauer
ton 10 run (se first :noten2) :dauer
make "noten bf :noten
```

```
make "noten2 bf :noten2
go "anf
end

to anfang
ton 65 run (se first :noten) :dauer
ton 66 run (se first :noten2) :dauer
make "noten bf :noten
make "noten2 bf :noten2
end
```

Als nächstes sollten wir einmal für den Parameter :dauer die Werte 50, 40, 30 und 20 einsetzen. Wie erklärt sich der hörbare Effekt (kurze Pausen nach jedem Ton)? Ganz einfach. Die Ausführung der Logo-Anweisungen ist insgesamt länger als der Nachschub an sound-Befehlen für den Tongenerator. Ist der letzte Ton ausgeklungen und steht kein weiterer Befehl unmittelbar an, so entsteht die hörbare Pause. Alle Musikexperimente werden hierunter etwas leiden, da Schneider-Logo langsamer als andere vergeichbare Logo-Versionen ist. Doch wir können uns immer noch ein bißchen helfen.

Der Tongenerator kann maximal vier sound-Befehle auf Vorrat in der Warteschlange speichern. Beim Start der Musik könnten wir also die beiden Kanäle synchronisieren und drei weitere Befehle auf Vorrat in die Warteschlange stellen. Diese Maßnahme bringt einen kleinen Fortschritt. Testen wir melodie2:

```
to melodie2 :noten :noten2 :dauer
anfang
repeat 3 [kern]
release 3
label "anf
if emptyp :noten [stop]
ton 17 run (se first :noten) :dauer
ton 10 run (se first :noten2) :dauer
make "noten bf :noten
make "noten2 bf :noten2
go "anf
end
```

Bei einer :dauer von 30 Einheiten treten bei den letzten Noten Zwangspausen ein. Eine weitere Verbesserung liefert die dritte Version, da wir jetzt sound direkt benutzen und nicht mittelbar über die Routine ton. Testen wir melodie3:

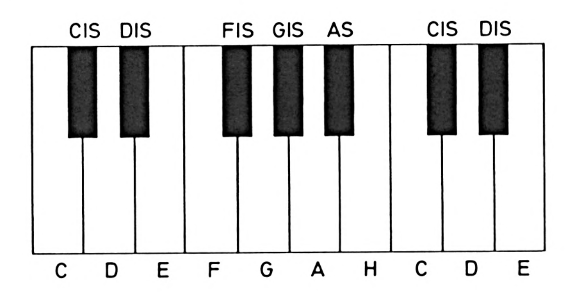
```
to kern
ton 17 run (se first :noten) :dauer
ton 10 run (se first :noten2) :dauer
make "noten bf :noten
make "noten2 bf :noten2
end

to melodie3 :noten :noten2 :dauer
anfang
repeat 3 [kern]
release 3
label "anf
if emptyp :noten [stop]
sound (se 17 run (se first :noten) :dauer)
sound (se 10 run (se first :noten2) :dauer)
make "noten bf :noten
make "noten2 bf :noten2
go "anf
end
```

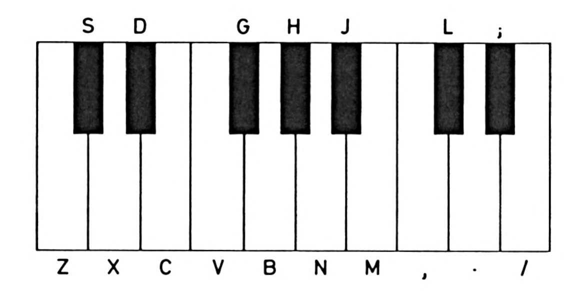
Wer noch mehr Musikstücke umsetzen will, möge sich ein Notenheft mit entsprechenden Liedern besorgen.

6.3 Ein Klavier muß her

Statt fertige Melodien zu produzieren, können wir ebenso aus unserem Schneider-Computer ein Klavier machen. Die Tasten werden dann eben zu Klaviertasten. Die Klaviertastatur einer Oktave sieht wie folgt aus.



Dieser Klaviertastatur sollen folgende Tasten des Keyboards entsprechen.



Die nachfolgende Programmidee ist einfach. Jede Taste hat den entsprechenden Ton gemäß obiger Vorgaben zu erzeugen. Die Z-Taste ergibt somit den Ton C, X den Ton D usw. Definieren wir die Töne der 2. Oktave, indem wir Programmnamen gemäß den Tasten erzeugen, die uns wie gehabt mit op die gewünschte Tonperiode liefern.

```
to z
op 478
end

to x
op 451
end

to v
op 358
end

to b
op 338
end

to n
op 284
end

to j
op 286
end

to m
op 301
end

to ,
op 239
end

to l
op 225
end

to .
op 213
end

to d
op 402
end
```

Jetzt müssen wir noch den Baustein erzeugen, der abhängig von der gedrückten Taste den Ton erzeugt. Hierzu benutzen wir die Logovokabel rc. rc ist die Abkürzung für read character und bedeutet «lies Zeichen».

pr rc

Geben wir diese Anweisung ein, so wartet der Computer so lange, bis eine Taste gedrückt wird. Der entsprechende Buchstabe oder Wert dieser Taste wird dann zur Eingabe von pr. In unserem Fall wird das rc zur Eingabe des Klavierprogramms.

```
to klavier :taste
if not memberp :taste tasten [stop]
ton 1 run (se :taste) 60
end

to tasten
op [z s x d c v g b h n j m , l .]
end
```

Mit der nachfolgenden Anweisungszeile können wir dann 1000 Klaviertasten drücken. Die Routine memberp entnehmen wir dem Anhang.

```
repeat 1000 [klavier rc]
```

Die nachfolgende Version vermeidet die bemerkbar werdenden Ausführungszeiten von Schneider-Logo. klavier1 benutzt die catch-Funktion, um falsche Tastendrücke abzufangen. Somit wird die Zeit zur Ausführung von memberp und der ersten Zeile in klavier vermieden.

```
to klavier1 :taste
catch "error [ton 1 run (se :taste) 60]
end
```

6.4 Forschungsaufträge

An dieser Stelle möge der Leser einmal die melodie-Programme mit Werten größer als 60 für den Parameter :dauer testen. Aber aufgepaßt, bitte vorher die Programme auf Diskette sichern. Bei längerer Tondauer werden die Warteschlangen zu groß, daß heißt, der zulässige Speicher läuft über. In diesem Falle «verklemmt» sich der CPC 464, nichts geht mehr. Aus diesem «Hang up» kommen wir nur durch erneutes Laden von Logo heraus. Dabei sind dann natürlich alle Programme des Arbeitsspeichers verloren. Hierzu setzen wir den Rechner zurück (reset)

mit CTRL-SHIFT-ESC und sind im BASIC-Modus. Dann tippen wir :CPM ein.

Auch beim Experimentieren mit der Rendezvous-Technik kann so etwas passieren. Also, erst die Programme sichern und dann testen.

Abschließend wollen wir noch ansatzweise die Befehle env und ent kennenlernen. env und/oder ent müssen vor Aufruf eines sound-Befehls ausgeführt worden sein. Man sollte genau das CPC-464-Handbuch studieren. In Logo müssen die Zahlenwerte in Form einer Liste eingegeben werden. Zuerst sehen wir uns den ton-Befehl mit seinen sieben Parametern an.

```
to ton :kanal :freq :dauer :laut :env :ent :gr
sound (se :kanal :freq :dauer :laut :env :ent :gr)
end
```

:kanal	Kanalstatus
:freq	Tonperiode (0 bis 4095)
:dauer	Tonlänge
:laut	Lautstärke (0 bis 6)
:env	Kennziffer für Hüllkurve der Lautstärke (bei 0 inaktiv)
:ent	Kennziffer für Tonhüllkurve (bei 0 inaktiv)
:gr	Kennziffer für Geräusch (0 bis 15)

Probieren wir der Reihe nach mittels unseres Klaviers die Wirkung aus.

```
env [1 5 1 14]
repeat 20 [klavier2 rc]
```

```
to klavier2 :taste
catch "error [ton 1 run (se :taste) 60 6 1 0 0]
end
```

In klavier2 sind die letzten zwei Parameter inaktiv, da sie jeweils den Wert 0 haben. Ändern wir die Hüllkurve systematisch und testen es mit unserem Klavier.

```
env [1 5 1 4]
repeat 20 [klavier2 rc]
```

Testen wir als nächstes den ent-Befehl mit beispielsweise:

```
ent [1 30 1 2]
repeat 20 [klavier3 rc 0]
```

```
to klavier3 :taste :gr
catch "error [ton 1 run (se :taste) 60 6 1 1 :gr]
end
```

Um ganz genau die Wirkung zu hören, sollten wir vielleicht in klavier3 den Wert für :env auf null setzen.

Als nächstes setzen wir die Geräuschfunktion ein, indem wir einen Wert für :gr im Bereich von 1 bis 15 wählen.

```
repeat 20 [klavier3 rc 3]
```

Logo-Kurzgrammatik

Logo ist äußerst gutmütig und erlaubt fast alles. Dennoch müssen wir einige Spielregeln einhalten. Logo erwartet von uns, daß wir beim Eintippen sorgfältig sind. Benutzen wir irgendwelche Logovokabeln, so müssen wir sie buchstabengetreu eintippen. Vor und nach dieser Vokabel darf kein anderer Buchstabe oder anderes Zeichen stehen außer dem Leerzeichen. Jede Logovokabel benötigt eine oder mehrere Eingaben. Wir müssen genau wissen oder nachschlagen, welche Eingaben und wie viele gefordert sind. Andernfalls gibt es eine Fehlermeldung; das heißt, die Anweisung wird nicht ausgeführt.

Objekte in Logo

Logo kennt letztlich nur zwei Dinge: das Wort und die Liste. Jedes Wort muß mit einem Anführungsstrich beginnen und kann aus einem oder mehreren Zeichen bestehen. Der Sonderfall eines Worts ist das Wort ohne Buchstaben, das leere Wort, das nur aus den Anführungsstrichen besteht. Die Zahlen gehören zu den Wörtern und können ausnahmsweise auch ohne Anführungsstriche benutzt werden. Eine Liste ist alles, was zwischen zwei eckigen Klammern steht. Die Elemente einer Liste können Wörter oder andere Listen innerhalb der Liste sein. Der Sonderfall einer Liste ist die leere Liste.

Innerhalb von Listen werden Wörter ohne Anführungsstriche aufgeführt. Beim Ausdrucken von Wörtern oder Listen werden die beiden äußeren eckigen Klammern und die Anführungsstriche nicht mitgedruckt.

Befehle und Operationen

Sowohl Logovokabeln (primitive) als auch Benutzerprogramme können die Eigenschaft eines Befehls oder einer Operation haben. Typische Logobefehle sind pr, make, repeat und die meisten Grafikbefehle (fd, bk, rt, lt usw.).

Operationen liefern Werte ab, die zu Eingaben bei anderen Logovokabeln oder Programmbausteinen werden können. Benutzerdefinierte Operationen müssen immer op enthalten. Die erste Operation, die wir kennengelernt haben, ist random gewesen.

Logovokabeln und Programme

Programme in Logo werden von uns definiert, indem die Befehle to und ed benutzt werden. Solche selbstdefinierten Bausteine bestehen aus Logovokabeln oder anderen selbstdefinierten Bausteinen. Der Benutzer kann seine selbstdefinierten Teilbausteine zu größeren Einheiten zusammensetzen. Logobefehle und Logooperationen können unmittelbar im Direktmodus ausgeführt werden. Programme werden mit ihrem Programmnamen aufgerufen, entweder durch direktes Eintippen des Programmnamens oder Aufruf aus einem anderen Programm heraus. Logovokabeln und eigene Bausteine können Eingaben (Variable) haben, nämlich Wörter oder Listen. Man spricht in diesem Fall besser von Parametern. Bei der Programmdefinition werden die Parameter alle namentlich in der Kopfzeile angeführt. Sie beginnen mit einem Doppelpunkt. Beim Programmaufruf müssen nach dem Programmnamen alle Parameter durch einen aktuellen Wert (ein Logoobjekt) befriedigt werden.

Leerzeichen, eckige Klammer, Anführungsstriche, Doppelpunkt

Diese vier Dinge sind für Logo lebenswichtig – ohne sie läuft nichts.

Jede Anweisungszeile mit ihren Elementen wird durch Leerzeichen in seine Einzelteile «zerhackt» (gegeneinander abgegrenzt). Das Leerzeichen ist ein Trennzeichen oder Trenner. Werden Leerzeichen vergessen, kann Logo die einzelnen Teile nicht mehr unterscheiden.

An eckigen Klammern erkennt Logo die Liste, am Anführungsstrich das Wort. Dinge ohne Anführungsstriche oder eckige Klammer sind eine Logovokabel oder ein Benutzerprogramm. Erhält eine Logovokabel versehentlich einen Anführungsstrich, ist das ein neues Wort und keine ausführbare Logovokabel mehr.

Der Inhalt einer Wertzuweisung wird durch direktes Voranstellen von Doppelpunkten (anstelle der Anführungsstriche) an den Variablennamen abgerufen.

Trennzeichen entschärfen

Neben dem Leerzeichen haben noch andere Sonderzeichen diese Trennereigenschaft (z. B. +, −, *, /, [,]). Sollen gelegentlich das Leerzeichen oder andere Zeichen mit Trennereigenschaft als normales Zeichen in Wörtern vorkommen, muß jedesmal mit einem vorangestellten «\» die Trennereigenschaft des nachfolgenden Trennzeichens aufgehoben werden. Der Schrägstrich hebt die Trennereigenschaft immer nur für ein einziges Zeichen auf!

Variable

Wertzuweisungen werden in Logo mit dem make-Befehl definiert. Auch Programmparameter können mit make innerhalb des jeweiligen Geltungsbereichs umdefiniert werden. Unterschieden werden globale und lokale Variable. Programmparameter sind immer Lokalvariable.

Anweisungszeilen in Logo

Eine vollständige Anweisungszeile beginnt in Logo mit einem Befehl (z. B. pr, repeat, fd, bk, rt...) und seinen Eingaben (Wörter, Listen, Operationen). Da Operationen (selbstdefinierte oder Logooperationen) mit ihren Eingaben ihrerseits Eingaben für andere Operationen liefern, können Logozeilen recht komplex wirken. Doch in dieser Verkettung von Operationen liegt eine der Stärken Logos. Man muß beim Interpretieren immer wissen, wieviel Eingaben jede Operation hat, um eine solche Zeile gedanklich korrekt aufzulösen. Das Ende einer Zeile wird durch das ENTER-Tastenzeichen angezeigt.

Typische Anfängerfehler

Die sechs klassischen Anfängerfehler sind:

1. «I don't know how to...»
   «Ich weiß nicht, wie ... gemacht werden soll»

```
?pr 3+4
7
?pr 3+4
I don't know how to pr3
```

Dieser Fehler entsteht hauptsächlich, wenn das Leerzeichen als Trenner vergessen wird. Der gleiche Fehler tritt ein, wenn eine Logovokabel oder ein Benutzerprogramm nicht buchstabengetreu angegeben wird.

2. «I don't know what to do with...»
   «Ich weiß nicht, was mit ⟨Objekt⟩geschehen soll»

```
?repeat 5 [count 1234567]
I don't know what to do with 7
```

Die Meldung entsteht, wenn ein Befehl in einer Anweisungszeile vergessen wird. Hier fehlt z. B. ein Druckbefehl vor dem count.

3. «...doesn't like...as input»
   «... mag ... nicht als Eingabe»

```
?pr 3+"b
+ doesn't like b as input
?pr first [ ]
first doesn't like [ ] as input
```

Diese Meldung entsteht, wenn Logovokabeln nicht zulässige Daten als Eingabe haben.

4. «not enough inputs to...»
   «Nicht genug Eingaben für ...»

```
?pr 3+
not enough inputs to +
```

Hier sind Eingaben zum Befriedigen aller notwendigen Parameter vergessen worden.

5. «...has no value in...»
   «... nicht definiert (ohne Inhalt) in ...»

```
?pr :vorname
vorname has no value
```

Diese Meldung tritt ein, wenn eine entsprechende Variable nicht definiert worden ist. Meistens ist die Fehlerquelle ein vertippter Variablenname, wenn die Parameter nicht buchstabengetreu im Programm eingesetzt werden.

Im nachfolgenden Beispiel werden alle bisherigen Fehler noch einmal gezeigt, bevor der Fehlertyp Nr. 6 demonstriert wird.

```
to adresse :name :str :ort
pr :namen
pr :str
pr:ort
end

?adressen [der fehlerteufel] [mistkaeferstr. 13] [1313 fehlerberg]
I don't know how to adressen

?adresse [der fehlerteufel] [mistkaeferstr. 13] [1313 fehlerberg]
namen has no value in adresse
```

Der Parameter :namen wird jetzt korrigiert und das Programm adresse erneut aufgerufen.

```
?adresse [der fehlerteufel] [mistkaeferstr. 13]
not enough inputs to adresse

?adresse [der fehlerteufel] [mistkaeferstr. 13] [1313 fehlerberg]
der fehlerteufel
mistkaeferstr. 13
I don't know how to pr:ort in adresse
```

Das vergessene Leerzeichen wird eingefügt bei pr :ort. Beim erneuten Testen wird jetzt eine eckige Klammer vergessen, so daß für Logo nur zwei Parameter statt drei befriedigt werden.

```
?adresse [der fehlerteufel [mistkaeferstr.13] [1313 fehlerberg]
not enough inputs to adresse
```

Endlich läuft das Programm ordnungsgemäß.

```
?adresse [der fehlerteufel] [mistkaeferstr.13] [1313 fehlerberg]
der fehlerteufel
mistkaeferstr.13
1313 fehlerberg
```

6. «...didn't output to...»
   «... hat nichts an ... geliefert»

Diese Meldung entsteht, wenn ein Programm zur Eingabe eines Parameters wird, aber keine Operation ist (op fehlt).

```
to doppelt :wort
pr word :wort :wort
end

?print doppelt "ha
haha
doppelt didn't output to print
```

Nützliche Programmbausteine

Nachfolgend sind einige Bausteine aufgelistet, die im Buch angeführt worden sind. Weitere sind neu und realisieren Vokabeln anderer Logoversionen, damit der Leser fortführende Logoliteratur benutzen und die angeführten Beispiele nachmachen kann.

Wer erläuternde Kommentare in seinen Bausteinen hinzufügen will, kann es mit dem Semikolonprogramm tun. Als Beispiel möge man sich farbauswahl ansehen.

```
to  ; :bemerkung
end

to farbauswahl :liste
 ; [:liste enthaelt farbzahlen]
 ; [die werte muessen im bereich 0 - 26 liegen]
if emptyp :liste [stop]
setpal 0 color first :liste
wait 30
farbauswahl bf :liste
end
```

Farbbefehle

Mit color wird eine Zahl zwischen 0 und 26 in die Farbverschlüsselung umgerechnet.

farbtester führt systematisch alle Farben vor.

```
to farbtester :nr
if :nr > 26 [stop]
ct
setpal 0 color :nr
wait 60
pr :nr
farbtester :nr + 1
end
```

```
to color :nr
if :nr < 3 [op se [0 0] trio :nr]
local "erg make "erg trio :nr
if :nr < 9 [op (se first :erg 0 last :erg)]
op (se first bf :erg first :erg last :erg)
end

to trio :dez
if :dez < 3 [op :dez]
op se trio int :dez / 3 rest :dez 3
end

to rest :zahl :durch
op (:zahl - :durch * int :zahl / :durch)
end
```

Arithmetik

Die vorige Operation rest entspricht dem remainder anderer Logoversionen.

Kreise und Kreisbögen

```
to kreisl :radius
repeat 36 [lt 5 fd 0.174532 * :radius lt 5]
end

to kreisr :radius
repeat 36 [rt 5 fd 0.174532 * :radius rt 5]
end

to bogenl :radius :grad
repeat int :grad / 10 [lt 5 fd 0.174532 * :radius lt 5]
end

to bogenr :radius :grad
repeat int :grad / 10 [rt 5 fd 0.174532 * :radius rt 5]
end
```

Wörter und Listen

```
to last :obj
op item count :obj :obj
end
```

```
to listp :obj
if or numberp :obj wordp :obj [op "FALSE]
op "TRUE
end

to thing :obj
op run se word ": :obj []
end

to memberp :obj :menge
label "start
if emptyp :menge [op "FALSE]
if :obj = first :menge [op "TRUE]
make "menge bf :menge
go "start
end

to member :element :objekt
label "start
if emptyp :objekt [op :objekt]
if :element = first :objekt [op :objekt]
make "objekt bf :objekt
go "start
end
```

Verwalten des Arbeitsspeichers
pons (po names) listet alle Globalvariablen.
erns (er names) löscht alle Globalvariablen.
pops (po procedures) listet alle Programme.

```
to pons
catch "error [po glist ".APV [stop]]
end

to erns
catch "error [ern glist ".APV]
end

to pops
catch "error [po glist ".DEF]
end
```

Definieren und Editieren

```
to copydef :neuname :altname
pprop :neuname ".DEF text :altname
end

to text :name
op gprop :name ".DEF
end

to define :name :was
pprop :name ".DEF :was
end
```

copydef erzeugt eine Programmkopie unter der Bezeichnung :neuname gemäß Programm :altname.

text liefert das Programm :name in Form einer Anweisungsliste.

define erzeugt ein Programm :name mit dem Inhalt der Anweisungsliste :was.

Grafikbefehle

```
to pos
op se first tf first bf tf
end

to xcor
op first tf
end

to ycor
op first bf tf
end

to setx :x
setpos se :x ycor
end

to sety :y
setpos se xcor :y
end
```

Übersicht der Schneider-Logo-Vokabeln

Beschreibung der Operanden der Logovokabeln

anweisungsliste
Eine durch Logo ausführbare Liste.

byte
Eine ganze Zahl im Bereich 0 bis 255.

datei
Ein aus Zeichen gebildeter Dateiname.

dauer
Eine ganze Zahl (0 bis 32767).

distanz
Eine reale Zahl.

eigsch
Die Bezeichnung (Name) einer Eigenschaft einer Eigenschaftsliste.

farbnummer
Eine dreielementige Liste aus Ziffern, wobei jede Ziffer die Werte 0, 1 oder 2 annehmen kann.

grad
Gradzahl eines Winkels.

hülle
Mehrere ganze Zahlen.

liste
Durch eckige Klammern eingeschlossene Logoobjekte.

kanalstatus
Eine ganze Zahl (0 bis 255).

kennzahl
Eine ganze Zahl (0 bis 15).

n, a, b
Eine beliebige Zahl.

name
Ein Wort, das ein Programm, eine Variable oder Eigenschaftsliste bezeichnet.

namen
Eine Namensliste, die Programme oder Variable bezeichnet.

objekt, obj
Ein Logo-Bestandteil (Wort, Liste oder Zahl).

nummer
Eine ganze Zahl 0 oder 1.

parameter
Wörter, die von einem Doppelpunkt

angeführt werden; sie werden im Zusammenhang mit to verwendet.

position, pos
Eine Liste zweier Zahlen, die die Position der Schildkröte oder des Cursors beschreibt.

präd
Ein logischer Ausdruck (Prädikat), dessen Auswertung entweder TRUE oder FALSE ergibt.

stiftnummer
Eine ganze Zahl 0, 1, 2 oder 3.

tonperiode
Eine ganze Zahl (0 bis 4095)

wort
Eine Buchstabenfolge, die keinen Zwischenraum enthalten darf.

zchn, Zeichen
Buchstabe, Ziffer oder Satzzeichen.

Beschreibung der Schneider-Logo-Vokabeln

Bemerkung: Einer Vokabel, die durch « * » gekennzeichnet ist, können beliebig viele Eingabevariable zugeordnet werden. Falls sich die Anzahl der verwendeten Variablen von der angegebenen unterscheidet, muß der gesamte Ausdruck in runden Klammern stehen. Die Vokabeln, die unter gewissen Bedingungen TRUE ergeben, führen zu FALSE, falls diese Bedingungen nicht erfüllt sind.

1. Schildkrötengrafik (Turtle-Grafik)

bk *distanz*
Die Schildkröte bewegt sich *distanz* Schritte zurück.

clean
Löscht den Grafikschirm, ohne den Zustand der Schildkröte zu beeinflussen.

cs
Löscht den Bildschirm, bewegt die Schildkröte zu Position 0,0 und setzt ihre Richtung zu 0.

dot *position*
Macht an der Bildschirmstelle *position* einen Grafikpunkt (dot).

fence
Begrenzt die Schildkröte auf den sichtbaren Bildschirmbereich.

fd *distanz*
Bewegt die Schildkröte *distanz* Schritte vorwärts.

ht
Macht die Schildkröte unsichtbar.

lt *grad*
Dreht die Schildkröte um *grad* Grad nach links.

pal *stiftnummer*
Liefert die Farbe des angegebenen Stifts in Form einer Liste.

pd
Setzt den Schreiber der Schildkröte auf die Zeichenfläche.

pe
Setzt den Radierstift ein.

px
Setzt den Invertierschreiber der Schildkröte auf die Zeichenfläche (dieser Schreiber löscht bereits gezeichnete Linien und zeichnet Linien auf noch freie Flächen).

pu
Hebt den Schreiber der Schildkröte von der Zeichenfläche ab.

rt *grad*
Dreht die Schildkröte um *grad* Grad nach rechts.

seth *grad*
Setzt die Richtung der Schildkröte auf den Wert *grad.*

setpal *stiftnummer farbnummer*
Bestimmt die gewünschte Farbe für den angegebenen Zeichenstift.

setpc *stiftnummer*
Setzt den Stift auf den Wert *stiftnummer* (0 bis 3).

setpos *position*
Setzt die Schildkröte auf die Stelle *position.*

sf
Liefert Schirmfakten in Form einer 5elementigen Liste (Hintergrundfarbe, Schirmstatus (fs, ss, ts), Anzahl Textzeilen im Splitmodus, Fensterstatus (fence, window, wrap) und den vertikalen Abbildungsmaßstab).

st
Macht die Schildkröte sichtbar.

tf
Ergibt eine 6elementige Liste über den Zustand der Schildkröte (x-Koordinate, y-Koordinate, Richtung, Stiftstatus (pn, pd, pe, px), Stiftfarbe und Sichtbarkeit (TRUE oder FALSE)).

window
Führt dazu, daß der Bildschirm einen Ausschnitt aus der Zeichenfläche der Schildkröte zeigt.

wrap
Führt dazu, daß die Schildkröte nach Verlassen des Bildschirms auf der gegenüberliegenden Seite wieder eingesetzt wird.

2. Wörter und Listen

ascii *zchn*
Ergibt den ASCII-Wert für *zchn.*

bf *obj*
Ergibt alle Elemente eines Objekts, bis auf das erste.

bl *obj*
Ergibt alle Elemente eines Objekts, bis auf das letzte.

char *n*
Ergibt das Zeichen, dessen ASCII-Code *n* ist.

count *objekt*
Liefert die Anzahl der Elemente von *objekt.*

emptyp *obj*
Ergibt TRUE, falls *obj* leer ist.

first *obj*
Ergibt das erste Element von *obj*.

fput *obj liste*
Fügt *obj* als erstes Element in *liste* ein.

item *n objekt*
Liefert das *n*-te Element von *objekt*.

* list *obj1 obj2*
Listet den Inhalt der genannten Variablen aus.

numberp *obj*
Ergibt TRUE, falls *obj* eine Zahl ist.

* se *obj1 obj2*
Falls es sich bei den Eingabevariablen um Worte handelt, wird aus diesen eine Liste gebildet. Handelt es sich bereits um Listen, wird aus allen Objekten der Listen eine neue Liste gebildet.

* word *word1 word2*
Fügt die Eingaben zu einem Wort zusammen.

wordp *obj*
Ergibt TRUE, falls *obj* ein Wort ist.

obj1 = *obj2*
Ergibt TRUE, falls *obj1* gleich *obj2* ist.

3. Variable

* local *name*
Macht die Variable namens *name* zur Lokalvariablen.

make *name obj*
Ordnet *name obj* zu.

namep *name*
Ergibt TRUE, falls *name* bereits einem Wert zugeordnet ist.

4. Arithmetische Operationen

cos *n*
Ergibt den Cosinus des Winkels *n* (*n* in Grad).

int *n*
Ergibt den ganzzahligen Anteil von *n*.

random *n*
Ergibt eine Zufallszahl zwischen 0 und $n-1$.

sin *n*
Ergibt den Sinus des Winkels *n*.

$a + b$
Ergibt $a + b$.

$a - b$
Ergibt $a - b$.

$a * b$
Ergibt $a * b$.

a / b
Ergibt a / b.

$a < b$
Ergibt TRUE, falls *a* kleiner als *b* ist.

$a > b$
Ergibt TRUE, falls *a* größer als *b* ist.

$a = b$
Ergibt TRUE, falls *a* gleich *b* ist.

5. Definieren und Editieren

ed *name(n)*
Startet den Editor mit dem/den genannten Programm(en).

end
Beendet eine durch to eingeleitete Definition.

to *name (parameter)*
Beginnt eine Programmdefinition.

6. Ablaufsteuerungen und Bedingungen

catch *name anweisungsliste*
Die *anweisungsliste* wird ausgeführt, falls der Befehl throw *name* ausgeführt wird.

co
Steht für continue und setzt den Programmlauf nach einem mit pause erzwungenen Halt fort. co kann nur vom Bediener über die Tastatur eingegeben werden.

error
Liefert eine sechselementige Liste, die den letzten Fehler beschreibt.

go *wort*
Sprungbefehl, der zu der angegebenen Programmzeile verzweigt, die durch *wort* markiert sein muß (vgl. label).

if präd *liste1 (liste2)*
Falls präd TRUE ergibt, wird *liste1* ausgeführt, sonst *liste2*

label *wort*
Markiert mit *wort* die Einsprungstelle innerhalb eines Programms für den Sprungbefehl go.

op *obj*
Übergibt die Ablaufsteuerung und den Wert *obj* an die aufrufende Routine.

pause
Hält ein laufendes Programm an (vgl. co).

repeat *n anweisungsliste*
Führt *anweisungsliste n*-mal aus.

run *anweisungsliste*
Führt *anweisungsliste* aus. Ergebnis ist, was von *anweisungsliste* ausgegeben wird.

stop
Unterbricht das laufende Teilprogramm (Prozedur) und übergibt die Ablaufkontrolle an das aufrufende Programm.

throw *wort*
Übergibt die Ablaufkontrolle an das korrespondierende catch *wort*.

wait *n*
Unterbricht für *n*-mal 0,22 Sekunden den Programmablauf.

7. Logische Operationen

*and *präd1 präd2*
Ergibt TRUE, falls alle Eingabevariablen TRUE sind.

not *präd*
Ergibt FALSE, wenn *präd* TRUE ist und umgekehrt.

*or *präd1 präd2*
Ergibt TRUE, sobald eine der Eingabevariablen TRUE ist.

8. Die Verbindung zur Außenwelt

buttonp *paddlenummer*
Ergibt TRUE, falls der Trigger des durch *paddlenummer* bezeichneten Paddles niedergedrückt ist.

bye
Verläßt Logo und ruft das Betriebssystem auf.

dir
Listet das Inhaltsverzeichnis der Diskette auf, abhängig vom zugewiesenen Laufwerk

ent *kennzahl hülle*
Ähnlich dem Befehl env kann eine Ton-Hüllkurve definiert werden.

env *kennzahl hülle*
Eine Verlaufsform für die Lautstärke wird durch *hülle* definiert. *kennzahl* und *hülle* bilden eine Liste. Einzelheiten im CPC-464-Handbuch.

keyp
Ergibt TRUE, falls eine Taste gedrückt, aber noch nicht eingelesen wurde.

paddle *paddlenummer*
Liefert den Zustand des durch *paddlenummer* bezeichneten Paddles.

*pr *obj*
Druckt *obj* gefolgt von einem Zeilenvorschub aus (Listen werden ohne äußere eckige Klammern ausgedruckt).

rc
Ergibt das von der Tastatur gelesene Zeichen. Falls kein Zeichen ansteht, wird das Programm bis zur Eingabe eines Zeichens angehalten.

release *n*
release gibt abhängig von der Zahl *n* (0 bis 7) einen oder mehrere gesperrte Tonkanäle frei (vergleiche Handbuch).

rl
Liefert die über Tastatur eingetippte Zeile als Liste.

rq
Liefert die über Tastatur eingetippte Zeile als Zeichenkette (Wort).

show *obj*
Druckt *obj* in eckigen Klammern aus, falls *obj* eine Liste ist.

sound *kanalstatus tonperiode dauer*
sound erzeugt einen Ton mit der Zeitdauer *dauer* und einer Frequenz abhängig von *tonperiode* auf einem der Kanäle A, B oder C, abhängig von *kanalstatus.* Ausführliche Einzelheiten im CPC-464-Handbuch. Neben den drei Pflichtparametern sind vier weitere wahlweise möglich. Alle Parameter bilden eine Liste.

⋆type *obj*
Druckt *obj* aus, ohne den Cursor an den Anfang der folgenden Zeile zu setzen.

9. Schirmorientierte Befehle

ct
Löscht den Textabschnitt des Bildschirms.

fs
Reserviert den ganzen Bildschirm für Grafik.

setsplit *n*
Legt die Anzahl der Textzeilen *n* im Splitmodus fest.

ss
Teilt den Bildschirm in Grafik- und Textbereich auf.

ts
Stellt den ganzen Schirm für Textausgabe zur Verfügung.

10. Verwaltung des Arbeitsspeichers

.contents
Listet alle dem System bekannten Dinge auf (Logovokabeln), Wertzuweisungen, Programme, Eigenschaftslisten usw.).

er *name(n)*
Löscht die genannten Programme.

ern *name(n)*
Löscht die genannten Variablen.

load *datei*
Lädt in den Arbeitsspeicher die Datei *datei.*

nodes
Ergibt die Anzahl der freien Nodes (Node = Speichereinheit).

po *name(n)*
Druckt die Definition der benannten Programme.

pots
Druckt die Namen aller definierten Programme.

recycle
Erzwingt eine Garbage-Collection.

save *datei*
Speichert den Inhalt des Arbeitsspeichers unter dem Namen *datei* auf der Diskette.

11. Eigenschaftslisten

glist *eigsch*
Listet alle Eigenschaftslisten des Arbeitsspeichers auf, die die Eigenschaft *eigsch* besitzen. (Hinweis: siehe im Anhang die Routinen pons, pops, erns).

gprop *name eigsch*
Liefert den Wert der Eigenschaft *eigsch* von *name.*

plist *name*
Liefert die Eigenschaftsliste *name.*

pprop *name eigsch objekt*
Ordnet der Eigenschaftsliste *name* für *eigsch* den Inhalt *objekt* zu.

remprop *name eigsch*
Entfernt aus einer Eigenschaftsliste *name* die angegebene Eigenschaft *eigsch.*

12. Assembler

.deposit *n byte*
Speichert das Datum *byte* in der Adresse, die durch *n* definiert wird.

.examine *n*
Ruft den Inhalt der Adresse, die durch *n* definiert wird, auf.

13. Spezialwörter

end
Signalisiert Logo, daß die Definition einer Funktion beendet ist.

ERRACT
Eine Systemvariable, die, falls TRUE, Logo bei auftretenden Fehlern pausieren läßt.

FALSE
Spezielle Eingabe für and, if, not, or oder Ergebnis von Prüfoperationen.

REDEFP
Systemvariable, die, falls TRUE, das Umdefinieren von Logo-Vokabeln erlaubt. (Nützlich zum Eindeutschen der Logovokabeln).

TOPLEVEL
Markierung für THROW, um die Kontrolle in den Toplevel-Zustand zu überführen.

TRUE
Spezielle Eingabe für AND, IF, NOT, OR oder Ergebnis einer Prüfoperation.

.APV
Systemeigenschaft zur Markierung von Globalvariablen.

.DEF
Systemeigenschaft zur Markierung von Benutzerprogrammen.

.PRM
Systemeigenschaft zur Markierung von Logovokabeln (primitives).

Kontrolltasten und Editierfunktionen

* kennzeichnet Tastenfunktionen, die innerhalb und außerhalb des Editors gelten.

\ (backslash)
Dieses Zeichen wird Zeichen mit Trennereigenschaft vorangestellt, um es diese Funktion zu nehmen.

* →
Bewegt den Cursor eine Stelle nach rechts.

* ←
Bewegt den Cursor eine Stelle nach links.

↑
Bewegt den Cursor eine Zeile nach oben.

↓
Bewegt den Cursor eine Zeile nach unten.

CAPS LOCK
Hält die Umschalttaste SHIFT in Dauerfunktion

COPY
Beendet den Editiermodus und kehrt in den Direktmodus zurück.

* CTRL-A
Bewegt den Cursor zum Anfang der Zeile.

CTRL-C
Wie COPY.

* CTRL-D
Löscht das Zeichen an der Cursor-Position.

* CTRL-E
Bewegt den Cursor zum Ende der Zeile.

CTRL-G
Wie ESC.

* CTRL-K
Löscht die Zeile von der Cursor-Position an bis zum Zeilenende.

CTRL-O
Erzeugt an der Cursor-Position eine Leerzeile.

CTRL-SHIFT-ESC
Setzt den Rechner zurück (reset).

CTRL-U
Führt im Editiermodus zu einem Vorschub auf die nächste Bildschirmseite.

CTRL-V
Zeigt die vorhergehende Bildschirmseite.

CTRL-Y
Ruft während des Editierens den Inhalt des Löschpuffers auf; außerhalb des Editors wird das zuletzt eingegebene Kommando ausgedruckt.

CTRL-Z
Entspricht dem Befehl pause.

DEL
Löscht das Zeichen links vom Cursor.

ENTER-Taste
Beendet die eingegebene Zeile und setzt den Cursor an den Anfang der nächsten Zeile.

ESC
Unterbricht alle laufenden Tätigkeiten. Im Editiermodus vorgenommene Änderungen werden ignoriert.

Stichwortverzeichnis

VOGEL-Computerbücher

Was der CPC 464 alles kann

von Martin Aschoff – Reihe HC – Mein Home-Computer

ISBN 3-8023-**0841**-7

Tips zum Programmieren in BASIC und Tricks zum Umgang mit dem Betriebssystem erhöhen den Nutzwert Ihres CPC 464.

Höhere Mathematik auf dem CPC 464

von Harald Baumgart – Reihe CHIP WISSEN

ISBN 3-8023-**0856**-5

Sie finden Programme zur Ausgleichsrechnung, zur Fehleranalyse und zur Funktionsbetrachtung – erweitert um die Problemkreise der höheren Mathematik.

Start in die Künstliche Intelligenz mit dem Schneider CPC 464

von Jeremy Vine – Reihe HC – Mein Home-Computer

ISBN 3-8023-**0863**-8

Folgende Techniken werden vermittelt: Aufbau von Datenbanken, Mustererkennung, wirkungsvolle Stringmanipulationen u. a. m.

Abenteuerspiele programmieren auf dem CPC 464

von A. J. Bradbury – Reihe HC – Mein Home-Computer

ISBN 3-8023-**0871**-9

Anhand von Beispielen wird die Entwicklung von Abenteuerspiel-Programmen demonstriert.

Superspiele und Utilities für CPC 464 und 664

von Jim Gregory – Reihe HC – Mein Home-Computer

ISBN 3-8023-**0870**-0

Eine Fülle von interessanten, spannenden Spielen.

aktiv computern

Willkommen bei Logo! Sie haben eine Diskette mit dem CP/M-Betriebssystem und Logo. Um die Neue DR.-Logo-Version optimal einsetzen zu können, sollten Sie diese Einführung benutzen. Hier wird mit Grafik, Text und Musik gespielt, gearbeitet, experimentiert. Lernen Sie das kleine Logo-Einmaleins in 12 Lektionen mit der Schildkrötengrafik. Große farbige Bildschirmfotos begleiten Sie durch die einzelnen Lernschritte. Das Buch verlangt nach praktischer Mitarbeit. Es gibt Ihnen Hilfen und Anregungen für eigenes Forschen. Dank seines bausteinartigen Konzepts kann jeder seine eigenen Teilprogramme erzeugen und neu zusammenstellen. So werden weitere interessante Einsatzbereiche erschlossen. Die Auflistung aller Schneider-Logo-Vokabeln im Anhang hilft Ihnen dabei.

VOGEL-BUCHVERLAG
WÜRZBURG

ISBN 3-8023-0867-0



Dietrich Senftleben
Start mit Logo auf dem CPC 464 und 664


AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
ACME
https://acpc.me/